НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмне забезпечення для зберігання та використання персональних документів»

Виконав:
студент IV курсу, групи КП-61
Згуровський Ярослав Юрійович _____

Керівник:
Старший викладач кафедри ПЗКС,
Гадиняк Руслан Анатолійович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Старший викладач кафедри ПМА,
Мальчиков Володимир Вікторович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

«ЗАТВЕРДЖУЮ»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Згуровському Ярославу Юрійовичу

1. Тема проєкту «Програмне забезпечення для зберігання та використання персональних документів», керівник проєкту Гадиняк Руслан Анатолійович, старший викладач, затверджені наказом по університету від «25» травня 2020 р. №1181-с.

2. Термін подання студентом проєкту «15» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - аналіз існуючих рішень;
   - обґрунтування вибору засобів реалізації;
   - структурно-алгоритмічна організація;
   - аналіз розробленого програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   - структура бази даних (креслення);
   - діаграма прецедентів (креслення);
   - блок-схема алгоритму обробки повідомлень із черги серверної частини (плакат);

– схема процесу шифрування персональних даних (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури програмного забезпечення	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну системи та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	01.03.2020	
7.	Програмна реалізація дипломного проєкту	01.04.2020	
8.	Тестування програмного забезпечення	12.04.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	25.04.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	02.05.2020	
11.	Підготовка графічної частини дипломного проєкту	09.05.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Ярослав ЗГУРОВСЬКИЙ

Керівник проєкту Руслан ГАДИНЯК

АНОТАЦІЯ

Даний дипломний проєкт присвячений розробці програмного забезпечення для зберігання та використання персональних документів з можливістю автозаповнення форм.

Розроблене програмне забезпечення являє собою крос-платформний мобільний застосунок, призначений для полегшення процесу зберігання документів в електронному вигляді та автоматизації їх використання у повсякденному житті. Функціональність програмних засобів дозволяє структурувати та виконувати основні операції над документами, такі як: створення, оновлення, видалення на мобільному пристрої та поширення для інших користувачів. За допомогою алгоритмів шифрування та хешування, система забезпечує захист персональних даних користувачів, оскільки хмарна база даних зберігає дані виключно в зашифрованому вигляді. Доступ до використання застосунку забезпечується лише після реєстрації, гостьовий доступ не передбачено. Сервіс автозаповнення дозволяє використовувати збережені дані для автоматизованого заповнення форм інших застосунків.

У даному дипломному проєкті розроблено: архітектуру мобільного застосунку та пов’язаної з ним серверної частини, сервіс для автозаповнення, алгоритми для шифрування персональних даних та безпечного поширення даних, а також графічні елементи та дизайн мобільного застосунку.

ABSTRACT

This diploma project is devoted to the development of software for storing and using personal documents with the ability to autocomplete forms.

Developed software is a cross-platform mobile application designed to facilitate the process of storing documents electronically and automate processes of using documents in everyday life. The functionality of the software provides an ability to structure and perform basic operations on documents, such as: creating, updating, deleting on a mobile device, and sharing with other users. With the help of encryption and hashing algorithms, the system protects the personal data of users, because the cloud database stores data only in an encrypted way. Access to the use of the application is granted only after registration, guest access is not allowed. The AutoFill service provides an ability to use saved data to automatically fill out forms in other applications.

While working on this project, the following artifacts have been developed: the architecture of the mobile application and the connected server-side part, the service for form autofill, the algorithms to encrypt, and to share sensitive personal data, the graphic elements and the design of the mobile application.

ДП.045440-01-90 Програмне забезпечення для зберігання та використання персональних документів. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Програмне забезпечення	5	
	для зберігання та		
	використання		
	персональних документів.		
	Технічне завдання		
ДП.045440-03-81	Програмне забезпечення	68	
	для зберігання та		
	використання		
	персональних документів.		
	Пояснювальна записка		
ДП.045440-04-51	Програмне забезпечення	4	
	для зберігання та		
	використання		
	персональних документів.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Програмне забезпечення	10	
	для зберігання та		
	використання		
	персональних документів.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Програмне забезпечення	1	
	для зберігання та		
	використання		
	персональних документів.		
	Структура бази даних.		
	ER-діаграма		
ДП.045440-07-99	Програмне забезпечення	1	
	для зберігання та		
	використання		
	персональних документів.		
	Функціональність		
	програмних засобів. UML-		
	діаграма прецедентів		
ДП.045440-08-98	Програмне забезпечення	1	
	для зберігання та		
	використання		
	персональних документів.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ТА ВИКОРИСТАННЯ ПЕРСОНАЛЬНИХ ДОКУМЕНТІВ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ярослав ЗГУРОВСЬКИЙ

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для зберігання та використання персональних документів.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості якості сервісу для зберігання персональних документів та використання персональних даних у повсякденних справах.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмна система повинна забезпечувати такі основні функції:

1. Можливість завантаження в сховище персональних документів в електронному вигляді.
2. Можливість скачування зі сховища персональних документів в електронному вигляді.
3. Можливість збереження та редагування персональних даних у сховищі.
4. Можливість автоматизованого заповнення форм персональними даними.

5. Можливість обміну персональними даними з іншими користувачами системи.
6. Можливість експорту вибраних персональних даних у певному форматі (JSON, XML тощо).

Додаткові вимоги:

1. Наявність адаптивного користувацького інтерфейсу.
2. Забезпечення конфіденційності та приватності персональних документів та даних.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1. Пояснювальна записка.
2. Програма та методика тестування.
3. Керівництво користувача.
4. Креслення:
   - «Структура бази даних. ER-діаграма»;
   - «Функціональність програмних засобів. UML-діаграма прецедентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту 16.11.2019
Розроблення та узгодження технічного завдання..... 01.12.2019
Розроблення структури програмного забезпечення 15.12.2019
Підготовка матеріалів першого розділу дипломного проєкту 29.12.2019
Розроблення дизайну системи та графічних елементів 20.01.2020
Підготовка матеріалів другого розділу дипломного проєкту..... 01.03.2020
Програмна реалізація дипломного проєкту..... 01.04.2020
Тестування програмного забезпечення..... 12.04.2020
Підготовка матеріалів третього розділу дипломного проєкту 25.04.2020
Підготовка матеріалів четвертого розділу дипломного проєкту ... 02.05.2020
Підготовка графічної частини дипломного проєкту 19.05.2020
Оформлення документації дипломного проєкту 26.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ТА ВИКОРИСТАННЯ ПЕРСОНАЛЬНИХ ДОКУМЕНТІВ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ярослав ЗГУРОВСЬКИЙ

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

HTTP cookie – це невеликий фрагмент даних, що відсилається сервером на браузер користувача, який браузер може зберегти і відсилати назад разом з новим запитом до даного сервера;

Indexed DB – низькорівневий API для клієнтського сховища великого об'єму структурованих даних, включаючи файли;

Solid – Social Linked Data, децентралізована модель зберігання і керування персональними даними;

HTTP – Hyper Text Transfer Protocol (протокол передачі гіпертекстових документів);

AES – Advanced Encryption Standard, також відомий під назвою Rijndael — симетричний алгоритм блочного шифрування (розмір блока 128 біт, ключ 128/192/256 біт), фіналіст конкурсу AES і прийнятий як американський стандарт шифрування урядом США;

RSA – криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел;

HTML – HyperText Markup Language, стандартизована мова розмітки документів в мережі;

JSON – JavaScript Object Notation, текстовий формат даних, заснований на JavaScript;

AWS – Amazon Web Services, компанія, яка надає послуги провайдера хмарних технологій;

FaaS – Function as a service, архітектурний шаблон, який передбачає можливість виклику серверного коду без необхідності управління серверами і серверним застосунком;

Amazon EC2 – Amazon Elastic Compute Cloud, розподілений сервіс хмарних обчислень;

IaaS – Infrastructure as a service, це модель обслуговування, в межах якої споживачу надається можливість керувати засобами обробки та

збереження, комунікаційними мережами, та іншими фундаментальними обчислювальними ресурсами;

DNS – Domain Name System, ієрархічна розподілена система перетворення імені хоста (комп'ютера або іншого мережевого пристрою) в IP-адресу;

Фаєрвол – пристрій або набір пристроїв, сконфігурованих, щоб допускати, відмовляти, шифрувати, пропускати через проксі весь комп'ютерний трафік між областями різної безпеки згідно з набором правил та інших критеріїв;

SSH – Secure Shell, протокол для віддаленої комунікації із пристроями;

DDOS-атака – Distributed Denial-of-service, напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена;

AWS Shield – це сервіс відкомпанії Amazon для захисту від DDoS-атак;

СУБД – Система Управління Базами Даних, інструментарій для управління базою даних та маніпуляції даних всередині неї;

ACID – Atomicity, Consistency, Isolation, Durability, це набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність;

ОС – операційна система;

S3 – Simple Storage Service, розподілене файлове сховище від компанії Amazon;

SQS – Simple Queue Service, розподілена черга від компанії Amazon;

SDK – Software Development Kit, набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи (програмної або програмно-апаратної).

ВСТУП

На сьогоднішній день є актуальною проблема зберігання персональних документів, що містять особисті дані. Зазвичай, люди зберігають власні документи, такі як паспорт, ідентифікаційний код, банківські виписки, рахунки, квитанції, податкові документи та інші, у паперовому вигляді. Цей спосіб є дуже ненадійним, оскільки існує ймовірність пошкодження документа внаслідок стихійного лиха чи іншої катастрофічної ситуації.

Якщо людині потрібно швидко надати інформацію, яка міститься в документі, або заповнити форму, що потребує цієї інформації, то їй потрібен час на пошук цього документу в місці, де він зберігається. Також є висока ймовірність внесення помилки у скопійовані дані, особливо при використанні мобільних пристроїв, таких як планшети, або смартфони. Наприклад, для того, що купити квитки на футбольний матч для компанії друзів необхідно особисті дані кожного. Таким чином, треба знайти власні документи, дістати з них потрібну інформацію та передати її тому, хто купує квитки. Цей процес не є дуже швидким та ефективним.

Саме тому створення програмної системи, яку можна буде використовувати у якості електронного сховища для персональних документів є актуальною задачею, оскільки таке сховище дозволить його користувачам надійно захищати та водночас ефективно використовувати дані з документів для повсякденних справ.

Даний дипломний проєкт присвячено розробці системи, що призначена для зберігання та використання персональних документів користувачів.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз моделей зберігання даних [1]

Однією з найбільших проблем, які необхідно вирішити під час реалізації програмної системи є проблема зберігання та забезпечення надійного захисту документів користувачів. Сучасні тенденції розвитку веб-технологій диктують умови та вимоги щодо того, як швидко ми можемо оперувати даними, наскільки ефективно їх використовувати та надійно захищати. Відповідно, постає задача у виборі найбільш надійної з точки зору безпеки та ефективної для зберігання та використання документів моделі управління даними.

Для аналізу було обрано наступні моделі:

- локального зберігання;
- віддаленого (хмарного) зберігання;
- децентралізована модель Solid;
- гібридна модель.

1.1.1. Модель локального зберігання

Додатки, що використовують локальну модель, зберігають дані на стороні клієнта розподіленої системи. Дані можуть бути зашифровані або відкриті. Інші частини системи (наприклад, віддалені сервери) лише отримують доступ до необхідних локальних даних за потреби. Прикладами веб-технологій, що реалізують дану модель є HTTP Cookies та Web Storage API, а також локальні бази даних (наприклад, IndexedDB API).

Основні переваги локальних сховищ:

- швидкий доступ до даних, що зберігаються у локальному сховищі;
- повний контроль над процесом зберігання даних, керуванням доступу та протоколами захисту інформації.

Основні недоліки:

- обмеження доступності лише з одного пристрою, що впливає на зручність використання та обміну;
- локальні резервні копії будуть знищені у випадку пошкодження фізичних пристроїв користувача.

1.1.2. Модель віддаленого (хмарного) зберігання

Хмарна модель передбачає зберігання даних користувача не на його пристрої, а на інших хостах мережі Інтернет.

Відповідно до дослідження статистичної служби “Eurostat”, останні кілька років зберігається тенденція до зростання у відсотках частки компаній, що використовують хмарні сервіси. Постачальники хмарних сховищ пропонують ефективність використання інфраструктури для зберігання даних, гнучкість та продуктивність у своїх послугах.

Основні переваги хмарних сховищ:

- можливість отримати дані з будь-якого місця за умови наявності Інтернет- з’єднання;
- можливість обміну даними, за умови доступу до ресурсів інших користувачів;
- можливість відновлення даних внаслідок непередбачуваної втрати за рахунок створення стратегії резервного копіювання постачальником.

Основні недоліки:

- неможливість отримання доступу до даних за відсутності Інтернет-з’єднання;
- за реалізацію безпеки і приватності персональних даних відповідає постачальник сервісу.

1.1.3. Децентралізована модель Solid

На даний момент активно ведеться розробка нової, так званої децентралізованої моделі зберігання і керування персональними даними – Solid (Social Linked Data). Це модель, що пропонує наступне – користувач володіє своїми власними даними, роблячи їх доступними для тих додатків, які хоче мати змогу використовувати.

Згідно набору принципів Solid, дані зберігаються в сховищах, що звуться “pods” (personal online data storage), які фізично розміщуються там, де обере користувач. Це може бути сторонній сервіс- постачальник таких сховищ, або локальний Solid сервер на пристрої користувача. Користувач може вирішити розподілити свої дані між кількома сховищами. Якщо Solid сховище автентифікує додаток та якщо користувач надав йому дозвіл на доступ до даних сховища, то постачальник надає застосунку дані, що зберігаються в конкретному сховищі.

1.1.4. Гібридна модель

Гібридна модель на основі локального та хмарного сховища, може задовольнити більшість потреб користувачів. Розгортання гібридного сховища надає можливості використання локального сховища для миттєвого доступу без Інтернет-з’єднання та резервного копіювання до хмари. У цьому випадку з’являються переваги швидкого та чутливого керування даними, одночасно дозволяючи обмін даними між різними користувачами та доступність даних з різних пристроїв.

1.1.5. Порівняння моделей зберігання даних

Для порівняння описаних моделей виділимо такі критерії:

- швидкість – наскільки швидко можна отримати доступ до даних;
- автономність – можливість використання без Інтернет-з’єднання;

- шифрування – можливість зашифрувати дані без передачі відкритих даних сторонньому ресурсу;
- обмін даними – можливість обмінюватись даними з іншими користувачами застосунку;
- резервні копії – можливість використання резервних копій при пошкодженні використовуваного клієнтом пристрою;
- стандарти – чи є готові стандарти, на які можна спиратись при розробці програмного забезпечення з використанням даної моделі.

Таблиця 1.1

Порівняння моделей зберігання даних

	Локальна	Хмарна	Solid	Гібридна
швидкість	висока	залежить від з'єднання	*	висока
автономність	+	–	*	+
шифрування	+	–	–	+
обмін даними	–	+	+	+
резервні копії	–	+	*	+
стандарти	+	+	–	+

* швидкість доступу, автономність та наявність резервних копій у моделі Solid залежать від розміщення сховища.

З точки зору темпів розвитку технологій та зручності використання сервісів, хмарні сховища безумовно мають перевагу перед локальними. Але в той же час існує найголовніший недолік: компанії-постачальники хмарних

сховищ можуть зловживати зберіганням персональних документів користувачів, порушуючи приватність та конфіденційність інформації.

Цю проблему частково може вирішити модель Solid, але, оскільки даний проєкт знаходиться ще на стадії розробки та постійних доопрацювань, пропонується відмовитись від його використання.

Отже, найбільш оптимальним рішенням з огляду на порівняльну характеристику моделей є проєктування системи, що використовує гібридну модель.

1.2. Огляд існуючих програмних рішень

1.2.1. Dropbox [2]

Dropbox – це сервіс зберігання файлів, що пропонує хмарний простір для сховища, можливість синхронізації файлів між різними пристроями і клієнтське програмне забезпечення. Онлайновий файловий хостинг Dropbox став свого часу першим загальнодоступним сервісом для хмарного зберігання і синхронізації даних. І досі, незважаючи на всі зусилля конкурентів, серед яких такі гіганти, як Google і Microsoft, залишається одним з найпопулярніших серед користувачів. Багато в чому ця популярність обумовлена гранично простою і зрозумілою концепцією сервісу. Приклад веб- інтерфейсу сервісу Dropbox зображено на рис. 1.1.

Основні характеристики та можливості сервісу Dropbox:

- будь-який файл, який користувач зберігає в Dropbox негайно стає доступним користувачу на всіх інших його пристроях: комп'ютерах, мобільних пристроях і через веб інтерфейс;
- 2 гігабайти хмарного сховища надаються безкоштовно кожному користувачеві;
- платні акаунти дають можливість зберігання до 100 гігабайт даних;
- файли завжди доступні через безпечне з'єднання з сайту Dropbox;
- клієнти для Windows, Mac, Linux, iOS, Android та Blackberry;

- сервіс можна використовувати, навіть якщо у користувача немає інтернет з'єднання;
- Dropbox передає файли для синхронізації по частинах, що робить його менш чутливим до перебоїв зв'язку;
- користувач може вручну задати ліміт використання сервісом Dropbox вашого інтернет-каналу, так що користувач не опиниться без зв'язку під час синхронізації з хмарним сховищем.

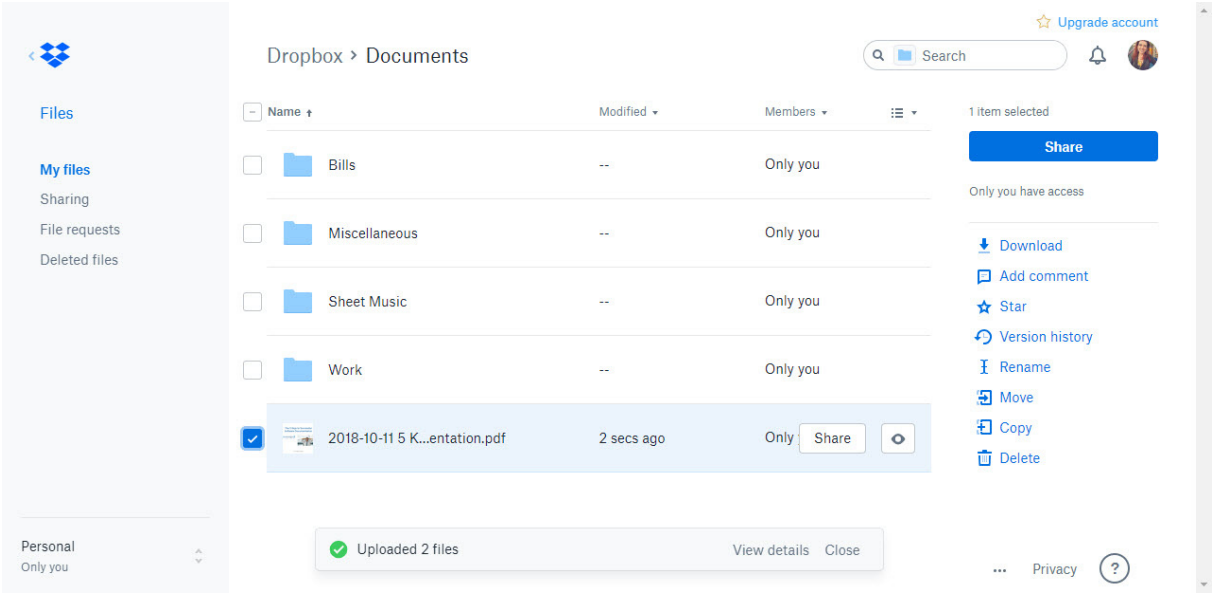


Рис. 1.1. Веб-інтерфейс сервісу Dropbox

1.2.2. Sync.com [3]

Sync.com була заснована в 2011 році в Канаді, як система, що перш за все підтримує конфіденційність. Sync.com має на меті полегшити користувачам доступ до своїх файлів та обмін ними практично з будь-якого комп’ютера, телефону чи мобільного пристрою – не відмовляючись від права на приватність і конфіденційність. Всі файли, завантажені в Sync.com, захищені 256-бітним AES-шифруванням. Це означає, що документ зашифрований у джерелі, і лише власник облікового запису – власне сам користувач – контролює ключ даних. Тож, хоча файли користувача знаходяться в Інтернеті, вони не мають сенсу для інших, оскільки для їх перегляду та використання потрібен ключ шифрування.

Ця властивість системи відома як конфіденційність з нульовими знаннями. Однак, оскільки Sync.com взагалі не містить ключі шифрування, це означає, що якщо користувач забуде свій пароль, то він втратить доступ до всіх даних, що зберігаються в даному сервісі.

Sync.com має клієнтські застосунки для десктопних операційних систем Windows та macOS, мобільних операційних систем Android та iOS та веб-інтерфейс.

З огляду на те, що шифрування файлів виконується на стороні клієнта, основним недоліком застосунку є швидкість роботи застосунку. Оскільки операції завантаження та скачування файлів з великим розміром можуть зайняти багато часу, це значно сповільнює швидкість взаємодії з документами для користувачів, котрі виконують дану операцію часто.

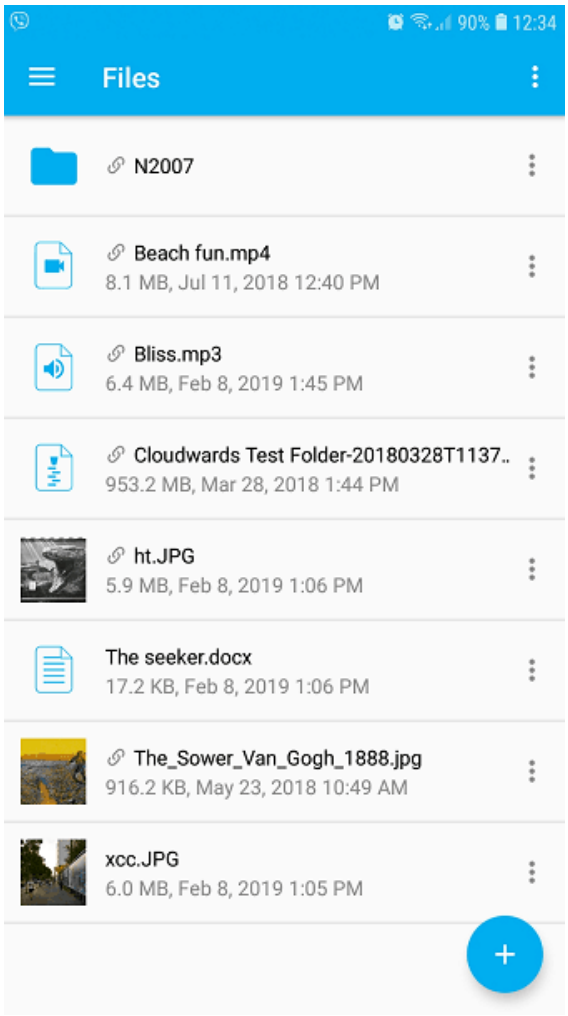


Рис. 1.2. Мобільний інтерфейс сервісу Sync.com

1.2.3. Tresorit [4]

Tresorit – це безпечний, потужний і швидкий хмарний сервіс зберігання даних, який входить до числа кращих в галузі. Tresorit – одне з найбільш захищених та приватних рішень для зберігання персональних документів завдяки шифруванню з методом нульового знання та підходу до синхронізації файлів і папок. Крім цього, він має багато інших функцій безпеки.

Основні можливості сервісу Tresorit:

- використання методу нульового знання для шифрування;
- повноцінне шифрування даних;
- двофакторна автентифікація і блокування пристроїв;
- підтримка версійності документів;
- обмін файлами за допомогою зашифрованих посилань;
- можливість встановлення терміну дії зашифрованого посилання;
- захист зашифрованих посилань паролем;
- можливість встановлення ліміту кількості завантажень для зашифрованих посилань;
- групи користувачів.

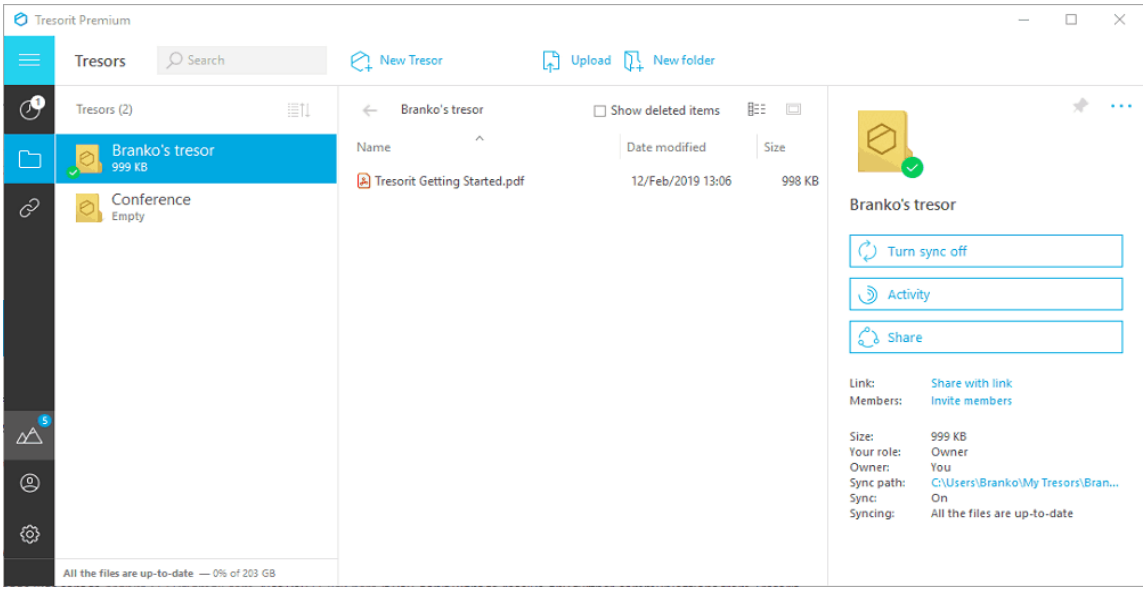


Рис. 1.3. Інтерфейс десктопного клієнта Tresorit

1.2.4. LastPass [5]

LastPass – програма, в основному призначена для зберігання та управління паролями користувачів, що реалізована у вигляді браузерних розширень, десктопних та мобільних застосунків. Паролі в LastPass зашифровані алгоритмом AES-256, зберігаються в «хмарі» і можуть бути синхронізовані між різними пристроями користувача. LastPass також має заповнювач форм, що дозволяє автоматизувати введення паролів, або заповнення форм іншими даними користувача.

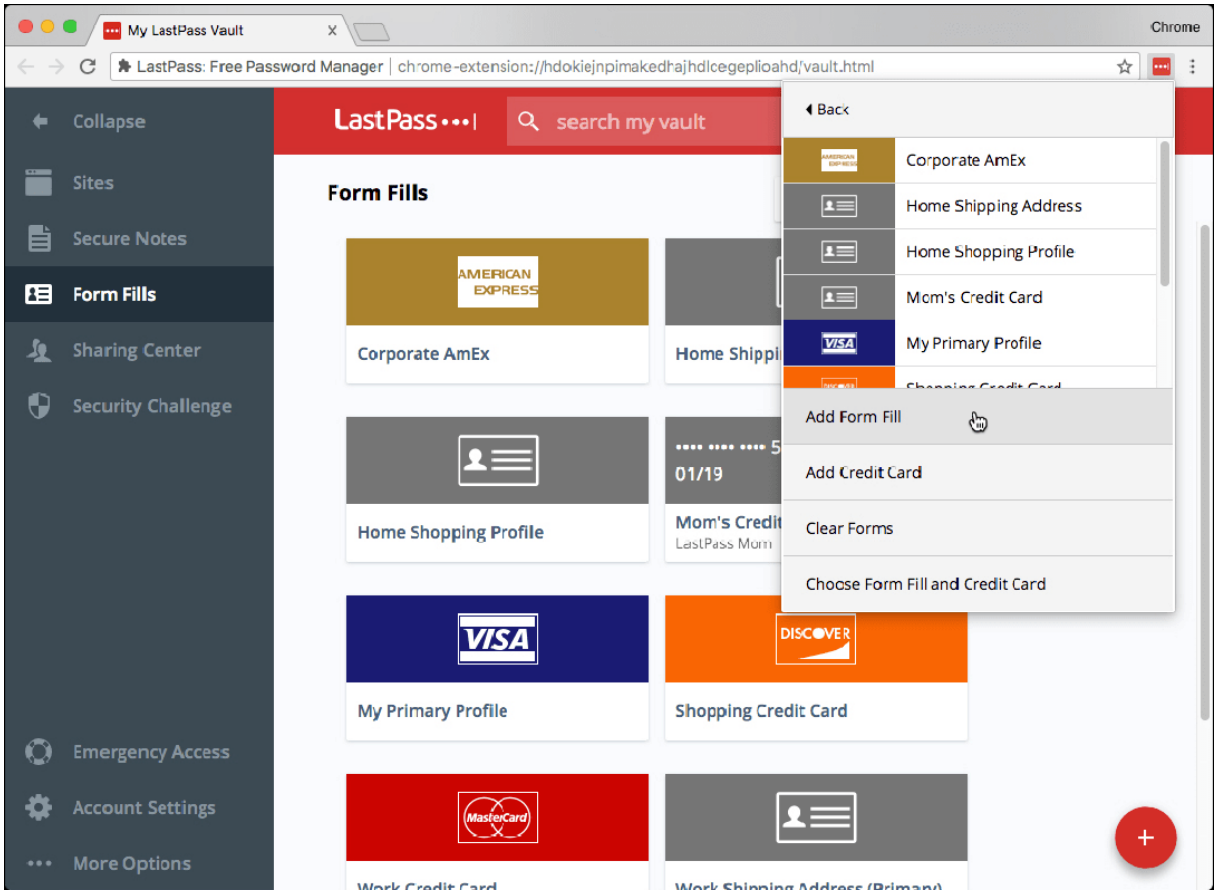


Рис. 1.4. Інтерфейс браузерного розширення LastPass

У LastPass є можливість створювати профілі, які містять, наприклад, виключно номер кредитної карти і ніякої додаткової інформації, або інші персональні дані. Коли користувач збирається заповнити веб-форму, він може вибрати профіль з персональними даними для автозаповнення. Якщо

є необхідність, то користувач може поділитися своїми профілями з іншими користувачами.

1.2.5. Результати аналізу існуючих рішень

Для порівняння було обрано найпопулярніші аналоги. Дані, що були отримані при детальному аналізі аналогів, їх переваг та недоліків, свідчать про те, що лише сервіс LastPass реалізує можливість автодоповнення форм даними, які зберігають сховища вищезгаданих сервісів.

При цьому, інші рішення реалізують зберігання документів та інших файлів користувачів. Dropbox надає багато можливостей для повсякденного використання, завдяки інтеграції з багатьма іншими сервісами. Також однією з найбільших його переваг є швидкість завантаження та скачування файлів з хмари. В той же час сервіси Sync.com та Tresorit роблять більший фокус на безпеці даних, гарантуючи приватність та конфіденційність інформації, завдяки методу нульового знання та використання шифрування на стороні клієнта. Вони відрізняються між собою кількома несуттєвими можливостями та характеристиками, такими як, наприклад, початковий обсяг сховища для файлів або ціна за використання сервісу.

Таким чином, результатом проведеного аналізу є виявлення того, що аналоги не в повній мірі реалізують вимоги до програмного продукту, що розробляється.

1.3. Постановка вимог до функціональності програмних засобів

В результаті аналізу моделей зберігання даних та наявних аналогів, їх переваг та недоліків та технічного завдання для дипломного проєкту було сформульовано список функціональних можливостей розроблюваного програмного забезпечення:

1. Система має надавати можливість реєстрації нового користувача та автентифікації існуючого.

2. Система має надавати можливість завантаження в хмарне сховище персональних документів в електронному вигляді.
3. Система має надавати можливість скачування з хмарного сховища персональних документів в електронному вигляді.
4. Система має надавати можливість збереження та редагування персональних даних у сховищі.
5. Система має надавати можливість автоматизованого заповнення форм персональними даними користувача.
6. Система має надавати користувачу можливість обміну персональними даними з іншими користувачами системи.
7. Система має надавати можливість експорту вибраних персональних даних у певному форматі (JSON, XML тощо).

Також було визначено нефункціональні вимоги до розроблюваної системи:

1. Система повинна використовувати шифрування на стороні клієнта, для забезпечення конфіденційності даних.
2. Система має надавати можливість користувачу використовувати документи та дані без доступу до мережі Інтернет.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір технології для розроблення серверної частини

Після аналізу сформульованих функціональних вимог програмної системи, було зроблено висновок щодо необхідності розбиття програмного додатку на два основні компоненти: серверну та клієнтську частини. Серверна частина відповідає за взаємодію з базою даних, яка розміщена в хмарі та обробку подій, що надходять з клієнтської частини. Клієнтська частина відповідає за взаємодію з користувачем та автозаповнення форм.

Для реалізації серверної частини було розглянуто наступні серверні платформи: AWS Lambda та Amazon EC2.

2.1.1. AWS Lambda

AWS Lambda – один із видів хмарного обчислювального ресурсу, що використовує модель FaaS, яка забезпечує можливість виклику серверного коду без необхідності управління сервером та серверним застосунком [6]. Основна ідея таких сервісів у тому, що серверна частина являє собою подійно-орієнтовану модель виконання комп’ютерних обчилень. Дана модель передбачає виконання програмного коду для обробки подій (поява нових повідомлень в черзі, HTTP-запит) без фіксування стану. Стан в даному випадку розуміється в контексті розгортання програмного застосунку. Це означає, що результат виконання даної серверної функції не залежить від стану пам’яті сервера, за виключенням параметрів, які були передані під час виклику функції. У такому випадку горизонтальне масштабування забезпечується хмарним провайдером, який може використовувати будь-який доступний йому в даний момент обчислювальний пристрій, що забезпечує необхідний рівень продуктивності. Це означає, що обчислювальні ресурси можуть масштабуватися та зменшуватися автоматично на основі запитів у режимі

реального часу. Також, як правило, час виконання функції примусово обмежується кількома хвилинами.

Основними перевагами використання AWS Lambda є:

- модель програмування: вхідним та вихідним параметром для Lambda обробників є JSON об’єкт, що може бути автоматично серіалізований або десеріалізований в структури даних на вибір розробника. Також функції можуть викликати інші API. Це означає, що розробнику не потрібно хвилюватися про багатопоточність або безпосередню обробку HTTP-запитів в коді, і це значно спрощує завдання розробки серверного програмного забезпечення;
- вартість: безсерверний ресурс, такий як AWS Lambda є більш вигідним за вартістю використання сервісом, ніж оренда, або покупка певної кількості серверів, яка зазвичай включає в себе періоди недозавантаженості, або навіть час простоювання без навантаження. До того ж, безсерверна архітектура означає, що розробникам і фахівцям з адміністрування не потрібно витрачати час, щоб налаштувати політики автоматичного масштабування – хмарний провайдер відповідальний за забезпечення достатніх обчислювальних можливостей.

До основних недоліків можна віднести:

- продуктивність: безсерверний процес, що не часто використовується, може мати більший час відгуку, аніж той, що безперервно працює в контейнері, або на виділеному сервері чи віртуальній машині. Це відбувається через те, що хмарний провайдер фактично зупиняє процес, якщо той не використовується. Наприклад, для розгортання нового середовища виконання Java необхідна значна кількість часу. Саме тому і виникає затримка при обробці запиту. Втім, не весь код сприйнятливий до затримок: наприклад, для операцій

пакетної обробки, запущених через cron, ці затримки будуть ледь відчутними;

- обмежені ресурси: AWS Lambda не підходить для певних типів ЕОМ, які потребують високопродуктивних обчислень, через обмеженість ресурсів, що встановлюють хмарні провайдери. Саме тому буде дешевше просто збільшити число серверів, необхідних в окремо взятий час;
- моніторинг та налаштування: діагностика проблем безсерверного коду, пов'язаних з надмірною витратою ресурсів або продуктивністю, є складнішою, в порівнянні з традиційним серверним кодом, тому що неможливо заглибитися і під’єднати профайлери та налагоджувачі в середовище виконання, де виконуваний код, зазвичай, не у відкритому вигляді. Отже, його продуктивні характеристики не можна точно відтворити в локальному середовищі.

2.1.2. Amazon EC2

Amazon EC2 забезпечує користувачів обчислювальними ресурсами за моделлю IaaS. За допомогою таких ресурсів можна створювати потужні сервери в хмарі [7].

IaaS надає наступні можливості використання хмарної інфраструктури:

- встановлення та використання програмного забезпечення, що включає в себе операційні системи, платформенне та прикладне програмне забезпечення;
- контроль над операційними системами, віртуальними системами зберігання даних та встановленими програмами;
- обмежений контроль за набором доступних мережевих сервісів (наприклад, фаєрволом, DNS).

Хмарним провадйером здійснюється управління основною фізичною і віртуальною інфраструктурою хмари, в тому числі серверів, типів використовуваних операційних систем, мережі та систем зберігання.

Основні переваги використання AWS EC2:

- уніфікована система управління: у разі використання EC2 користувач використовує єдиний інтерфейс управління, замість багатьох систем, що вимагають контролю та моніторингу. Таким чином, можна підвищити ефективність та надійність приватних і публічних хмарних середовищ;
- взаємна сумісність: EC2 сервіс підтримує велику кількість хмарних, фізичних та віртуальних платформ.

Основним недоліком є налаштування та адміністрування: розробникам або адміністраторам треба конфігурувати операційні системи та встановлювати додаткове програмне забезпечення для того, щоб мати змогу розгортати сервери для власних потреб.

2.1.3. Порівняльний аналіз AWS Lambda та AWS EC2

Проведено порівняльний аналіз двох технологій за наступними характеристиками:

- розгортання та управління середовищем: При використанні AWS Lambda потрібно лише завантажити код, що має виконуватись під час виклику, у вигляді архіву. Не потрібно перейматись про фізичні ресурси, або оркестрацію контейнерів, це повністю автоматизовано. Налаштування EC2 включає вхід у систему через SSH та встановлення Apache вручну та здійснення клонування git. Також потрібно встановити та налаштувати все необхідне програмне забезпечення [8];
- безпека: Безсерверна архітектура дозволяє уникнути роботи з оновленням ОС. Lambda забезпечує більшу гнучкість робочих процесів, але в той же час це збільшує поверхню атак. Однак

порушення вразливості рідше трапляються з огляду на характеристики Lambda, як функцій без стану. Крім того, функції автоматично масштабуються, що забезпечує хороший захист від DDOS-атак, але, в той же час, збільшується вартість використання сервісу. З EC2 доведеться піклуватися про рівень безпеки на рівні екземпляра сервера. Рівень безпеки визначає та контролює трафік, дозволений для зв'язку з кожним екземпляром. Кожен екземпляр може мати кілька рівнів захисту, що диктує дозволений вхідний трафік через певні протоколи, такі як TCP, UDP, ICMP і т.д. Створення політик захисту, які матимуть правильні дозволи, є складним процесом. Якщо розглянути DDOS-атаки, то доведеться додатково налаштовувати інші сервіси AWS Shield, або зробити це вручну, використовуючи лоад балансер для масштабування під атакою або обмеження кількості запитів на хвилину від конкретного ідентифікатора;

- час на виконання роботи: Відповідно до офіційної документації, час роботи AWS Lambda становить 15 хвилин. Це обмежує тип завдань, з якими може справлятися Lambda. Тривалі функції та складні завдання не дуже підходять. Деякі таймаути виникають через помилки відних параметрів, або коли відбувається комунікація через зовнішні сервери, яка займає занадто багато часу або відповідає на функції з неправильними даними. У порівнянні з AWS Lambda, EC2 точно може працювати з тривалими завданнями, оскільки екземпляри серверів доступні для різних типів вимог з різною конфігурацією;
- можливість підключення додаткових бібліотек: Зовнішні бібліотеки неминучі для будь-якого проєкту, і зокрема для AWS Lambda. Якщо відбувається робота важкими функціями обробки, такими як обробка зображень, конвертація відео, то виникає залежність від сторонніх бібліотек. У Lambda

максимальний розмір пакету, що виконується – 50 Мб. В той же час є можливість завантажувати залежності під час виконання. Для зберігання таких файлів є обмеження в 512 Мб. Крім того, чим більша кількість залежностей, тим більше часу знадобиться для виконання функції. Управління залежностями в EC2 не є великою проблемою, оскільки не має обмежень, коли мова йде про тимчасове зберігання. Хоча слід враховувати розмір програмних пакетів та відповідний процесор екземпляра. Це пов'язано з тим, що процесор може зазнати навантаження, якщо він не налаштований на нього;

- масштабованість: Масштабованість є однією з головних переваг Lambda, оскільки більшість деталей є автоматичною та керується AWS. Сервіс динамічно масштабується у відповідь на збільшення трафіку. На момент сплеску збільшується кількість одночасно виконуваних функцій. Максимальна кількість одночасних запитів на сьогодні обмежено 1000. З EC2 доведеться вручну налаштувати масштабування на відміну від Lambda. Однак позитивною стороною є те, що можна зробити свою масштабованість безпроблемною. Крім того, існують групи масштабування EC2 для автоматичного масштабування. Ці групи допомагають забезпечити наявність достатньої кількості екземплярів для обробки навантаження. Але для цього потрібно докласти додаткових зусиль для початкового налаштування;
- доступність: AWS Lambda постійно доступні для виклику та запускаються автоматично, залежно від вимог тригерів події. З EC2 не має переваги у вигляді запуску за наявністю запиту. Групи автоматичного масштабування допомагають у масштабуванні, але врешті-решт, доведеться тримати принаймні один запущений та постійно працюючий екземпляр серверу;

- період затримки до початку роботи: “Холодний старт” вважається однією з найбільших проблем безсерверних функцій, керованих подіями. Він виникає, під час запуску функції, яка була неактивною протягом значного часу. Ця затримка виникає через час, необхідний для надання вибраному контейнеру виконання, а потім виконання своїх функцій, а отже, це збільшує затримку функції. Це може зайняти більше 5 секунд, що не дає змоги гарантувати запуск менш ніж за 1 секунду функції від настання події-тригеру. “Холодні старти” не трапляються в екземплярах EC2, як правило, якщо не відбувається запуск нового контейнера. Процес створення нових екземплярів відрізняється високою затримкою і потрібно більше часу для готовності, ніж для існуючих екземплярів, оскільки вони, як правило, виконують код під час першого запуску. Наприклад, якщо зупинити екземпляр і запустити його заново, екземпляр буде доступний досить швидко.

В результаті проведеного аналізу було встановлено, що обидва сервіси задовольняють вимоги, що висуваються до серверної частини: високий рівень масштабованості та можливість забезпечення безпеки обробки персональних даних. З огляду на простоту налаштування та вищий рівень автоматизації розробки обрано AWS Lambda у якості технології для розробки серверного обробника.

2.2. Вибір СУБД для хмарного сховища

Після аналізу початкових вимог, було сформовано наступні вимоги до СУБД, що використовуватиметься при побудові серверної частини програмного забезпечення:

- реляційна модель даних;
- підтримка індексів;
- відповідність властивостей ACID;

- можливість розгортання на віддаленому сервері.

Розглянуто дві найпопулярніші СУБД, що існують на сьогоднішній день та проведено порівняльний аналіз за вказаними вимогами.

2.2.1. PostgreSQL

PostgreSQL – одна з найпопулярніших реляційних СУБД з відкритим вихідним кодом, яка часто використовується при створенні різних типів застосунків [9]. Ця база даних є однією із перших реляційних СУБД, що з'явилися на ринку. PostgreSQL дозволяє оперувати як структурованими, так і неструктурованими даними. Вона підтримується більшістю сучасних операційних систем, включаючи Windows, Linux, MacOS. Також PostgreSQL має зручні вбудовані інструменти для імпорту даних з інших СУБД.

Основними перевагами PostgreSQL є:

- повна відповідність вимогам ACID;
- клієнт-серверна архітектура: сервер може підтримувати велику кількість одночасних окремих підключень від додатків-клієнтів;
- потужний повнотекстовий пошук з підтримкою регулярних виразів.
- великі можливості масштабування бази даних;
- можливість зберігати XML, JSON та інші унікальні типи даних;
- підтримує великі можливості управління транзакціями;
- надає можливість виконання коду, що написаний деякими мовами програмування, прямо на сервері СУБД.

Основними недоліками PostgreSQL є:

- неповна підтримка стандарту ANSI SQL;
- помітне падіння швидкодії при групових операціях;
- порівняна складність встановлення та конфігурації для новачків.

2.2.2. MySQL

MySQL – реляційна СУБД, що розроблюється та безкоштовно розповсюджується компанією Oracle [10]. Дана СУБД написана мовами C та C++. Також вона є сумісною з усіма основними операційними системами.

Основними особливостями СУБД MySQL є [11]:

- повна відповідність вимогам ACID;
- клієнт-серверна архітектура: для взаємодії із базою даних, до сервера MySQL під'єднуються незалежні клієнти-додатки;
- простота використання: підтримка всіх наявних в SQL типів даних;
- швидкість вивчення: незначна відмінність синтаксису MySQL від стандарту SQL;
- підтримка основних рівнів ізоляції транзакцій та функцій їх відкату;
- розповсюдженість: для абсолютної більшості мов програмування існують бібліотеки для використання MySQL;
- невисока ціна розгортання та утримання відносно конкурентів.

До основних недоліків MySQL відносять:

- складність масштабування: налаштування ефективного горизонтального масштабування є нетривіальною задачею;
- складність самостійного налаштування операцій, що наявні «з коробки» в багатьох інших СУБД;
- можливість падіння швидкодії на надскладних запитах до БД: MySQL є популярною серед розробників веб-орієнтованих застосунків через велику кількість спеціальних функцій, таких як, наприклад, типи даних HTML.

2.2.3. Висновок

В результаті проведеного аналізу було встановлено, що обидві розглянуті СУБД повністю або частково відповідають вищезгаданим

вимогам до СУБД. Водночас, при розробленні серверної частини системи для використання у якості хмарної бази дних було обрано PostgreSQL, оскільки вона має більші можливості для роботи з транзакціями та надає переваги повнотекстового пошуку.

2.3. Вибір середовища виконання для серверної частини

Проведено порівняльний аналіз та обрано середовище виконання та відповідну мову програмування для розробки серверних Lambda обробників. Для порівняння розглянуто дві найпопулярніші строго-типізовані, об'єктно-орієнтовані мови програмування: C#, що виконується на платформі .NET Core та Java.

2.3.1. .NET Core

.NET Core – це крос-платформна модульна платформа з відкритим вихідним кодом для розробки програмного забезпечення. .NET Core дозволяє створювати застосунки з підтримкою різних операційних систем: Windows, Linux і macOS. Високопродуктивне середовище виконання для Windows Server і Linux робить .NET Core найефективнішою веб-платформою в тестах TechEmpower [12].

Продуктивність і масштабованість особливо важливі для архітектур мікросервісів. Середовище .NET Core дозволяє зменшити кількість серверів і віртуальних машин, необхідних для програмної системи. В кінцевому рахунку, це сприяє скороченню витрат на архітектуру і розміщення. Спрощені можливості і модульний принцип середовища .NET Core роблять його оптимальним для контейнерів.

Особливості використання .NET Core [13]:

- підтримка NuGet: як і всі інші мови, що підтримуються Lambda, .NET Core отримує підтримку модулів через NuGet, що значно полегшує процес написання коду розробникам;

- висока продуктивність: .NET Core має більш стійкий результат продуктивності, ніж динамічні мови програмування Node.js або Python;
- швидке виконання: у порівнянні з Java, .NET Core має швидший час виконання.

2.3.2. Java

Java – об'єктно-орієнтована мова програмування, а також потужна платформа розробки, яка використовується для створення багаторівневих програм та швидкодіючих серверних додатків. Java – це портативна мова, що означає те, що Java-програма може бути розроблена на будь-якому пристрої, скомпільована в стандартний байт-код і, як очікується, буде працювати на будь-якому іншому пристрої з віртуальною машиною Java (JVM). Java є досить швидкою мовою програмування. Компоненти, включаючи бібліотеки, фреймворки (наприклад, Hibernate), Java API, середовище виконання Java, Java плагіни та віртуальну машину Java (JVM), роблять її надійним інструментом для систем корпоративного рівня.

Особливості використання Java в AWS Lambda [14]:

- надійні та добре перевірені бібліотеки, що полегшують процес написання коду розробникам;
- прогнозована ефективність: у Java час “холодного старту” є доволі високим, але водночас розробник може легко передбачити потреби в пам'яті своїх функцій.

2.3.3. Висновок

З огляду на вищу продуктивність під час виконання коду оберемо середовище виконання .NET Core та, відповідно мову програмування C# для реалізації серверного Lambda обробника.

2.4. Вибір платформи для розроблення клієнтської частини

2.4.1. Розширення веб-браузера

Розширення браузера – це програма, яка доповнює функціональність браузера [15]. Також вона здатна вбудовуватися в хмарний софт за допомогою браузера і таким чином розширити його функціональність. Такі програми називать add-on (доповнення), plug-in (плагін), або browser extensions (розширення браузера). На даний момент, вони підтримуються майже всіма веб-браузерами, але найпопулярнішими є розширення для Mozilla Firefox та Google Chrome.

Для розроблення браузерних розширень використовуються стандартні веб-технології: JavaScript, HTML та CSS, а також деякі спеціальні JavaScript API.

Плагін дозволяє вбудувати розроблені елементи в хмарний сервіс з метою інтеграції, створення мікросервіса, додаткових можливостей для SaaS, або просто як додатковий інструмент. Серед іншого, доповнення дозволяють змінювати зовнішній вигляд або вміст певних сайтів.

Існує кілька типів браузерних розширень:

- інтеграційні розширення – надають можливість використати хмарний сервіс вмонтувавши в нього одну або кілька своїх кнопок. Таким чином можна налагодити процес обміну даними між різними внутрішніми системами та отримати додаткову функціональність до хмарного програмного забезпечення;
- додаткові можливості SaaS в будь-якій вкладці браузера – фонові сервіси різного призначення: перекладачі, підказки, календарі, перевірки, помічники та інші;
- мікросервіси – це невеликі застосунки, що взаємодіють зі сторінкою. Вони потрібні для сервісів, які має сенс запускати в середовищі іншого сайту/веб-додатка. Даний мікросервіс здатний зчитувати інформацію з сайту, на якому його викликали, і тим самим істотно спростити користувачеві взаємодію. Це

можуть бути будь-які сервіси скріншотів, заміток або відео. Також вони можуть не просто зберігати дані, а відразу завантажувати в хмарний сервіс, або відправляти електронною поштою.

Основні переваги браузерних розширень:

- швидкість доступу, зручність і зрозумілість використання;
- крос-платформність – здатність працювати на будь-якій платформі, де є веб-браузер;
- можливість об'єднання власної системи та хмарних сервісів у комплексну систему;

Основні недоліки розширень веб-браузерів:

- необхідність періодичних оновлень розширення у випадку оновлення браузера або сервісу;
- необхідність розробки окремої версії розширення під кожен браузер.

2.4.2. Мобільні застосунки

На сьогоднішній день мобільні застосунки займають все більше і більше часу в житті людини. Згідно статистичних досліджень, людина кожного дня проводить близько чотирьох годин на день, використовуючи месенджери, соціальні мережі та інші, наприклад, розважальні застосунки [16]. Тому пропонується розглянути можливість розробки клієнтської частини програмного забезпечення у якості мобільного додатку.

Основні переваги мобільних застосунків [17]:

- доступ до системних ресурсів мобільного пристрою (камера, GPS), звідси можливість забезпечити більше функціональності;
- можливість використання функціональності програм без підключення до мережі Інтернет;
- швидкість роботи мобільних застосунків є досить високою;

- можливість використання push-сповіщень, що забезпечують кращу взаємодію з користувачем.

Основні недоліки:

- необхідно забезпечити сумісність з кількома операційними системами: Android, IOS;
- складніший процес розробки, у порівнянні з веб-застосунками.

2.4.3. Висновок

Мобільний застосунок можна вважати більш зручним у повсякденному використанні для зберігання персональних документів, оскільки дана платформа забезпечує виконання більшості функціональних вимог для клієнтської частини застосунку.

2.5. Вибір технології для розроблення мобільного застосунку

Для вибору технології для реалізації мобільного застосунку розглянемо два сучасних фреймворки для розробки мобільних додатків: Flutter та React Native.

2.5.1. Flutter

Flutter – відносно новий фреймворк для розробки крос-платформних мобільних та веб-додатків з відкритим вихідним кодом від компанії Google. Він використовує для розробки мову програмування Dart [18]. Це швидка, об’єктно-орієнтована мова програмування, яку можна порівнювати у використанні з популярними об’єктно-орієнтованими мовами програмування, такими як C# та Java. Архітектура фреймворку базується на популярному реактивному стилі програмування. Flutter пропонує для використання власний набір віджетів, які генеруються за допомогою високопродуктивного двигуна для рендерингу. Саме завдяки системі віджетів, інтерфейс користувача виглядає інтуїтивно-зрозумілим та привабливим з точки зору функціональності та використання.

2.5.2. React Native

React Native – це фреймворк з відкритим кодом, що використовується для розробки мобільних застосунків. Його основою стала популярна JavaScript бібліотека для розробки веб-застосунків у браузері – React. React Native є крос-платформним фреймворком, що означає одночасну розробку додатків для Android та IOS. Насправді, “під капотом” React Native є таким собі мостом, що викликає вбудовані API візуалізації в Java (для Android) або Objective-C (для iOS). За рахунок цього, додаток відображається за допомогою реальних компонентів мобільного інтерфейсу, і виглядає як будь-який інший мобільний додаток. React Native також відкриває інтерфейси JavaScript для API мобільних платформ, тому додатки, що реалізовані з використанням даного фреймворку можуть отримувати доступ до функцій платформи мобільного пристрою, таких як місцезнаходження користувача чи камера телефону.

2.5.3. Порівняльний аналіз

Для вибору технології розробки мобільного застосунку було проведено порівняльний аналіз фреймворків за наступними критеріями:

- швидкість та продуктивність у використанні: Flutter забезпечує більшу продуктивність за рахунок використання графічної бібліотеки Skia [19]. Завдяки цьому інтерфейс користувача переробляється щоразу, коли змінюється стан об’єкту, що відтворює модель даних. Більшість завдань виконується на GPU (графічний процесор), тому інтерфейс Flutter є плавним і забезпечує 60 кадрів в секунду;
- швидкість реалізації застосунку: Flutter є однозначним фаворитом у характеристиці швидкості реалізації застосунку, оскільки він має “з коробки” велику кількість віджетів, з яких можна за короткий проміжок часу компонувати готові для використання мобільні застосунки. Також існує велика кількість

плагінів, що можна підключити та використати для реалізації основної функціональності додатку. React Native забезпечує розробника лише базовими компонентами;

- компоненти користувацького інтерфейсу: Вбудовані віджети Flutter сумісні з обома мобільними платформами: Android та IOS. Це забезпечує однакове відображення додатку на різних мобільних операційних системах. Такі ж самі властивості мають компоненти React Native. Але Flutter гарантує зворотню сумісність розроблених додатків навіть із старішими версіями мобільних операційних систем.

2.5.4. Висновок

З огляду на порівняльний аналіз фреймворків, було обрано Flutter у якості технології для розробки мобільного застосунку, оскільки він забезпечує високу продуктивність у використанні та швидкість розробки за рахунок більш широкого вибору вбудованих віджетів.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

3.1. Загальна структура системи

Загальна структура розробленої системи зображена на рис. 3.1.

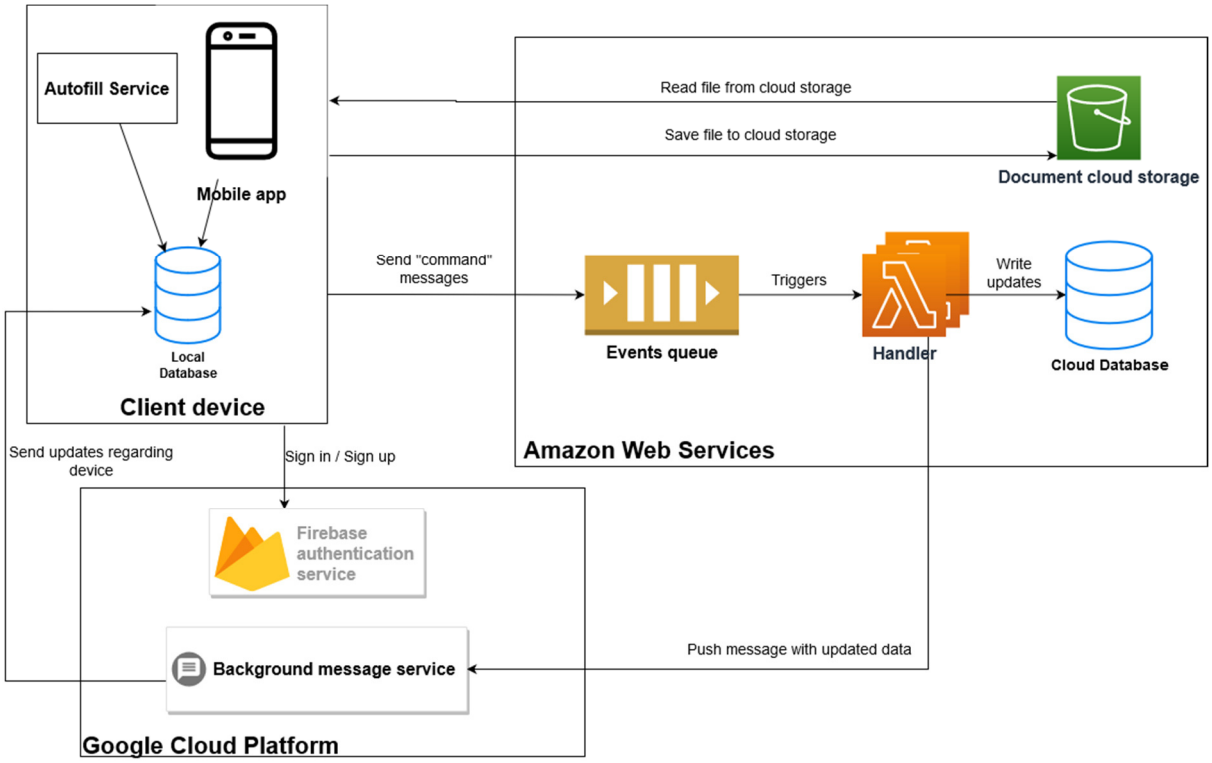


Рис. 3.1. Структурна схема програмної системи

Розроблене програмне забезпечення реалізоване у вигляді крос-платформного мобільного застосунку та складається з клієнтської частини, серверної частини та компонент, що використовують сервіси хмарних провайдерів. На схемі наведено основні шляхи взаємодії ключових елементів системи, а саме: взаємодія клієнтського застосунку з серверним обробником за допомогою черги повідомлень та хмарним сховищем документів, взаємодія обробника з хмарною базою даних та сервісом для відправки повідомлень на мобільні пристрої, взаємодія клієнтського застосунку з сервісом автентифікації.

Далі детально розглянуто компоненти та сервіси, що використовуються для їх реалізації.

3.1.1. Сервіс автентифікації

В якості сервісу автентифікації використано Firebase Authentication Service [20]. Даний хмарний сервіс надає бекенд-сервіси автентифікації, прості та зрозумілі SDK для розробників та бібліотеки для інтерфейсу користувача. В даній програмній системі використовується автентифікація користувачів за допомогою їхньої електронної адреси та паролю. SDK надає можливості для створення та управління користувачами, що використовують електронну адресу та пароль для входу та реєстрації. Для реєстрації нового користувача модуль автентифікації клієнтського застосунку викликає сервіс і передає йому параметри, що користувач ввів для реєстрації: електронна адреса та захешований пароль. У відповідь сервіс надсилає інформацію, щодо успішності реєстрації та унікальний ідентифікатор користувача.

3.1.2. Черга для обробки подій

Для даного компоненту використовується сервіс Amazon Simple Queue Service (далі SQS) [21]. SQS надає можливість роз'єднати розподілені компоненти системи за допомогою черги повідомлень (рис. 3.2).

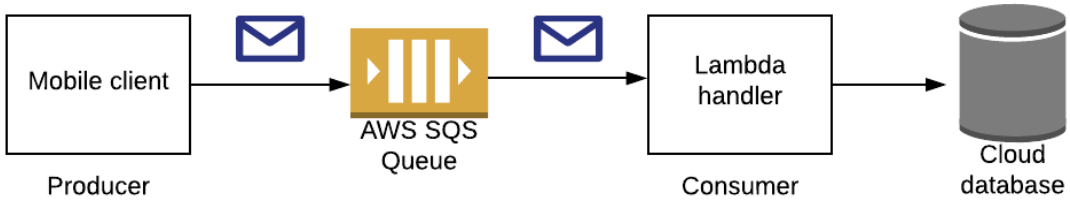


Рис. 3.2. Схема взаємодії компонентів з чергою повідомлень

Таким чином, клієнтський застосунок, використовуючи протокол HTTPS, надсилає повідомлення серверному обробнику для оновлення

хмарної бази даних та оновлень локальних баз даних інших користувачів у разі поширення доступу до персонального документу.

3.1.3. Серверний Lambda обробник

Для обробника подій із черги повідомлень використовується сервіс AWS Lambda [22]. Даний сервіс опитує чергу на наявність повідомлень у ній, та синхронно викликає код функції Lambda обробника з подією, що містить повідомлення. Сервіс зчитує повідомлення пакетами (кількість повідомлень в одному пакеті конфігурується) та викликає функцію один раз для кожного пакету. Коли функція зчитала повідомлення, вони все ще залишаються в черзі, але стають недоступними для зчитування на певний проміжок часу, що зветься тайм-аут видимості (visibility timeout) (рис. 3.3).

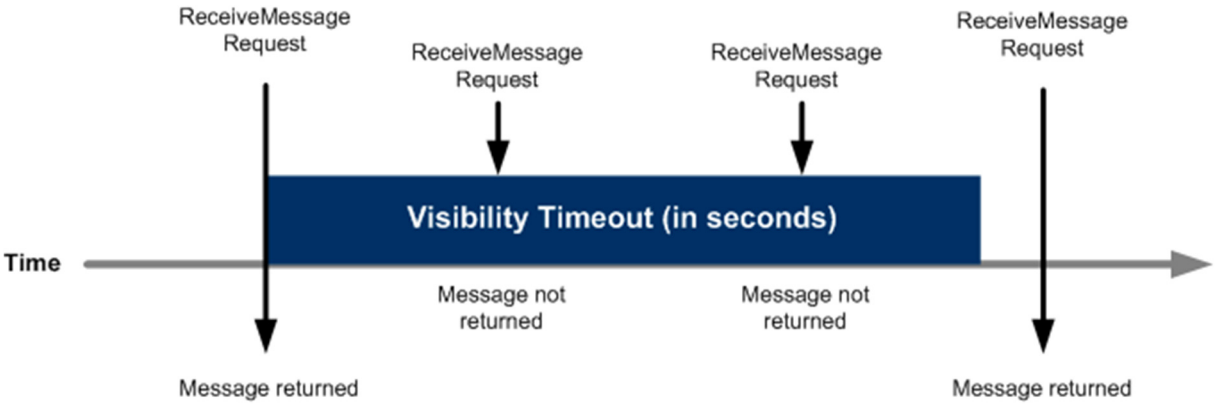


Рис. 3.3. Часова діаграма стану повідомлення у черзі

Якщо функція успішно обробила пакет повідомлень, то Lambda сервіс автоматично видаляє їх із черги. У випадку, якщо функція повертає у відповідь помилку, або не відповідає, то повідомлення з пакету знову стають видимими у черзі (див. Додаток 1. Блок-схема алгоритму обробки повідомлень із черги серверної частини).

Кожне повідомлення містить ключ автентифікації користувача, що надіслав повідомлення у чергу. Даний ключ порівнюється з тим ключем, що зберігається для кожного користувача в хмарній базі даних і визначається

чи валідним є дане повідомлення для подальшої обробки. У випадку успішної автентифікації повідомлення з черги, хмарний обробник робить зміни в базі даних та, за потреби, надсилає повідомлення для іншого клієнта, використовуючи сервіс для відправки повідомлень на мобільні застосунки.

Lambda обробник розділений на наступні модулі:

- модуль для взаємодії з базою даних;
- модуль для відправки повідомлень;
- модуль шифрування та хешування;
- модуль автентифікації запитів.

3.1.3. Сервіс для відправки повідомлень

У якості сервісу для відправки повідомлень з даними для клієнтського застосунку використовується Firebase Cloud Messaging сервіс (далі FCM). FCM дає можливість надсилати повідомлення на мобільні застосунки з даними, у вигляді пар ключ-значення (рис. 3.4) [23].

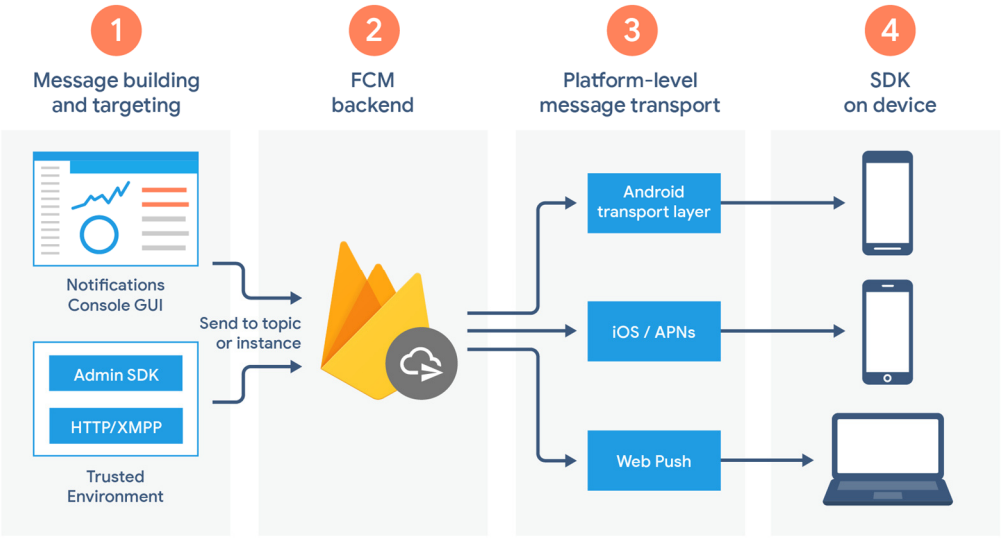


Рис. 3.4. Схема роботи сервісу FCM

Коли користувач хоче поділитись документом з іншим користувачем, то він надсилає відповідну подію на сервер. Далі обробник створює запис в

базі даних, формує нове повідомлення з документом та ідентифікатором пристрою і надсилає його до FCM. FCM сервіс приймає цей запит, генерує унікальний ідентифікатор повідомлення з іншими метаданими, та пересилає повідомлення на клієнтський пристрій, використовуючи специфічний для платформи пристрою транспортний канал.

Клієнтський пристрій має обробити дане повідомлення, викликавши функцію-колбек, яка створить запис про документ вже в локальній базі даних користувача, з яким поділились документом.

3.1.4. Сховище документів

Для хмарного сховища документів, використовується Amazon Simple Storage Service (Amazon S3) – сервіс, що дозволяє завантажувати та зберігати файли [24]. Для цього було створено S3 bucket – абстракція, що слугує саме тим ресурсом, де зберігаються файли, що завантажуються. Цей сервіс має точки доступу API, за допомогою яких клієнтський застосунок може завантажувати та скачувати файли для синхронізації з хмарним сховищем.

3.1.5. Хмарна база даних

Для хмарної бази даних використовується сервіс Amazon Relational Database Service (далі Amazon RDS). Даний сервіс дозволяє створювати екземпляри баз даних та управляти ними. У якості СУБД використовується PostgreSQL. Структуру хмарної бази даних детально описано в підрозділі 3.2.

3.1.6. Локальна база даних

Для зберігання інформації, що стосується лише окремого користувача використовується локальна база даних. У якості СУБД використовується SQLite, оскільки це файлова база даних, яка може зберігати структуровані реляційні дані на мобільному пристрої. Локальна база даних, має таку ж

саму структуру бази даних, як і хмарна, але на відміну від віддаленого сховища, не містить таблиць, що зберігають інформацію про користувачів та пристрої користувачів.

3.1.7. Сервіс автозаповнення

Для того, щоб мати можливість заповнювати форми даних в інших застосунках, розроблено модуль, що реалізує можливості сервісу автозаповнення. Для реалізації автозаповнення використано Autofill framework (рис. 3.5).

Autofill framework складається з наступних компонент [25]:

- Autofill service – зберігають інформацію для заповнення форм інших застосунків;
- Autofill client – застосунок, що містить форми, які мають бути заповнені;
- Android / IOS system – операційна система, яка визначає робочий процес і забезпечує інфраструктуру, яка змушує сервіс та клієнтський застосунок працювати разом.

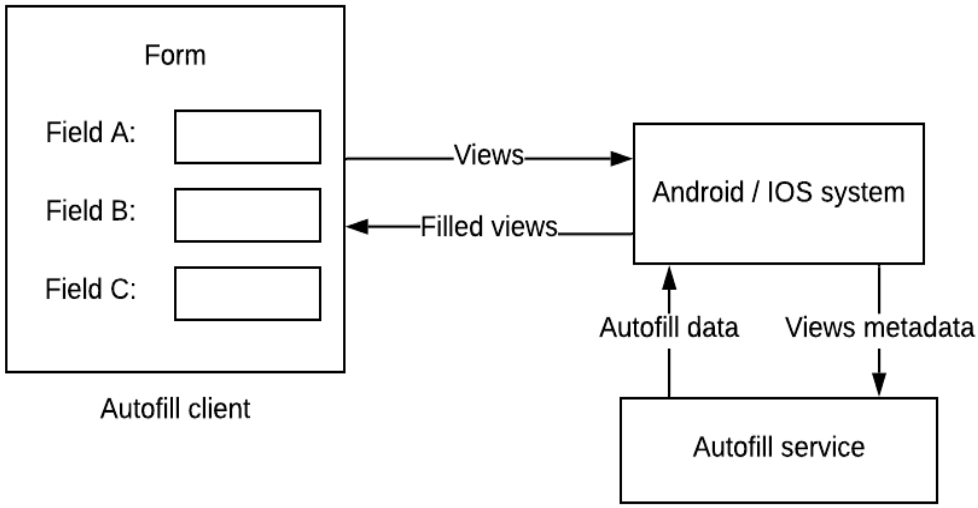


Рис. 3.5. Схема взаємодії компонентів Autofill framework

У якості ресурсу даних, реалізований модуль використовує локальну базу даних клієнтського пристрою, в якій зберігається інформація про поля документів та їх дані для заповнення форм інших застосунків. При наведенні фокусу на поле форми, сервіс отримує запит з метаданими поля для заповнення.

3.1.8. Клієнтський застосунок

Архітектура клієнтського застосунку побудована з використанням шаблону BLoC (Business Logic Component), що створений компанією Google. BLoC використовується для управління складним станом додатку та ґрунтується на реактивній парадигмі (рис. 3.6). Даний патерн допомагає розділити рівень презентації даних від рівня бізнес-логіки.

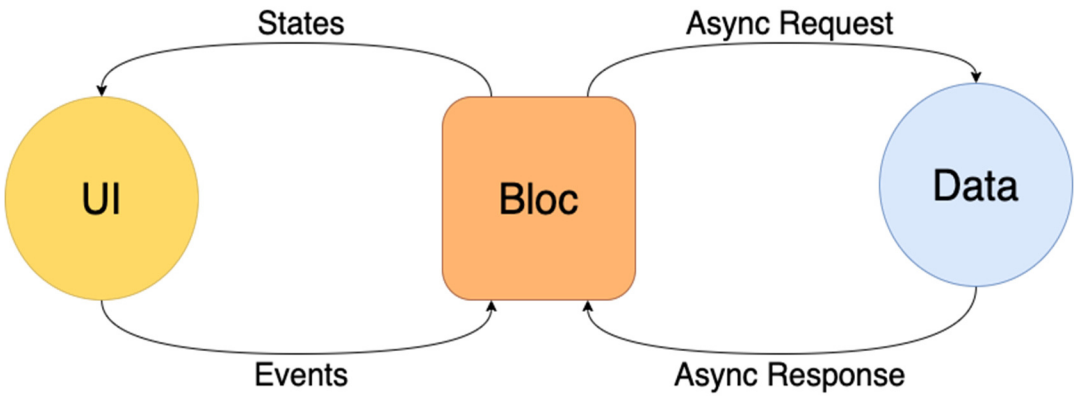


Рис. 3.6. Схема принципу роботи патерну BLoC

Основна ідея патерну BLoC полягає в тому, що додаток розбивається на модулі, що реалізують певну частину бізнес-логіки. Кожен модуль має одну або кілька «труб», які є деяким вхідним потоком для агрегування подій ззовні, тобто дій користувача. У якості вихідних даних виступає «потік», який визначає асинхронний формат даних для віджетів застосунку. Щоб скористатися модулем на рівні керування, застосовують StreamBuilder, який керує потоком даних і автоматично вирішує проблеми підписки та перемальовування дочірнього дерева віджетів.

Використання BLoC в чистому вигляді – досить складна робота для розробника, оскільки необхідно використовувати бібліотеку RxDart для маніпуляції з потоками, вручну відписуватися від потоків, інакше є загроза виникнення серйозних витоків пам'яті в роботі з великими додатками [26]. З метою вирішення цих проблем вирішено використати бібліотеку bloc, яка по максимуму спрощує використання цього шаблону і надає зручний API для керування станом з можливістю легкого тестування модулів. Важливою перевагою цього пакета є можливість автогенерації коду за допомогою плагінів для найбільш популярних IDE (IntelliJ, VS Code). Таким чином, майже не витрачається час на написання зайвого коду та існує гнучкість в зміні логіки компонентів.

Основні переваги використання даного підходу:

- зручне API при роботі з потоками, що дозволяє їх легко групувати, поєднувати і трансформувати;
- легкість в тестуванні стану зі сторонніми ефектами за рахунок вбудованого в API мови Dart тестування потоків;
- об’єднання логіки, шляхом групування її в одному місці;
- мінімальна кількість відмальовувань віджетів, завдяки використанню StreamBuilder.

З недоліків можна виділити лише завищену складність для розробників-початківців, оскільки необхідно певний час, для того, щоб вникнути в суть роботи потоків.

Клієнтський застосунок розділено на наступні BLoC:

- Authentication – реалізує логіку, що визначає стан автентифікації користувача. Має наступні події: AppStarted, LoggedIn, LoggedOut;
- DocumentFields – реалізує логіку для форми віджету заповнення полів документу. Включає подію EmptyFieldAdded;
- Items – реалізує основну логіку роботи з папками та документами. Включає в себе наступні події: ItemsLoaded,

DocumentAdded, DocumentDeleted, DocumentUpdated, DocumentLoaded, DocumentShared, FolderAdded, FolderUpdated, FolderDeleted;

- Login – реалізує логіку для процесу входу користувача в застосунок, включаючи валідацію введених даних та взаємодію з кнопками. Має наступні події: LoginWithCredentialsPressed, Submitted, PasswordChanged, EmailChanged;
- Register – реалізує логіку процесу реєстрації користувача в застосунку. Включає наступні події: Submitted, PasswordChanged, EmailChanged;
- Tab – реалізує логіку відображення документів при переході між різними табами. Включає в себе подію TabUpdated.

Діаграма компонентів для рівня бізнес-логіки клієнтського застосунку зображена на рис. 3.7.

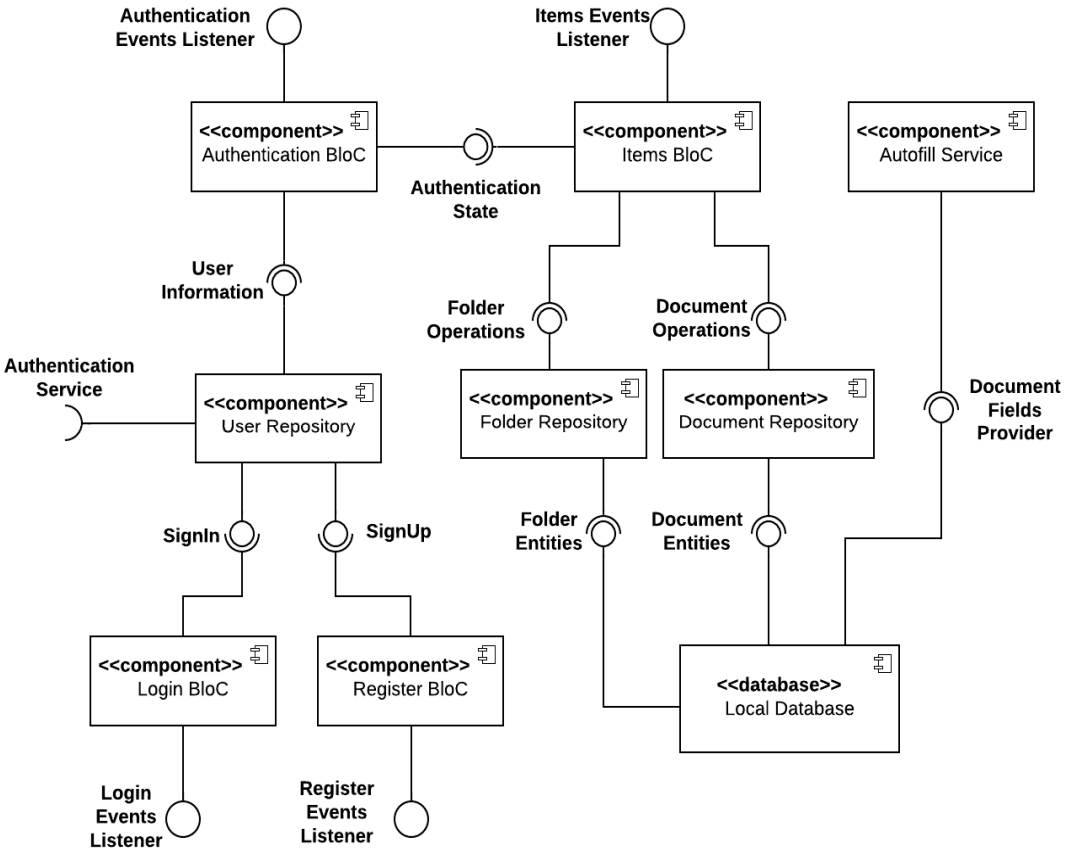


Рис. 3.7. Діаграма компонентів бізнес-логіки клієнтського застосунку

3.2. Опис структури даних системи

Враховуючи всі умови та вимоги до розроблюваного програмного забезпечення складено наступну схему бази даних системи (рис. 3.8).

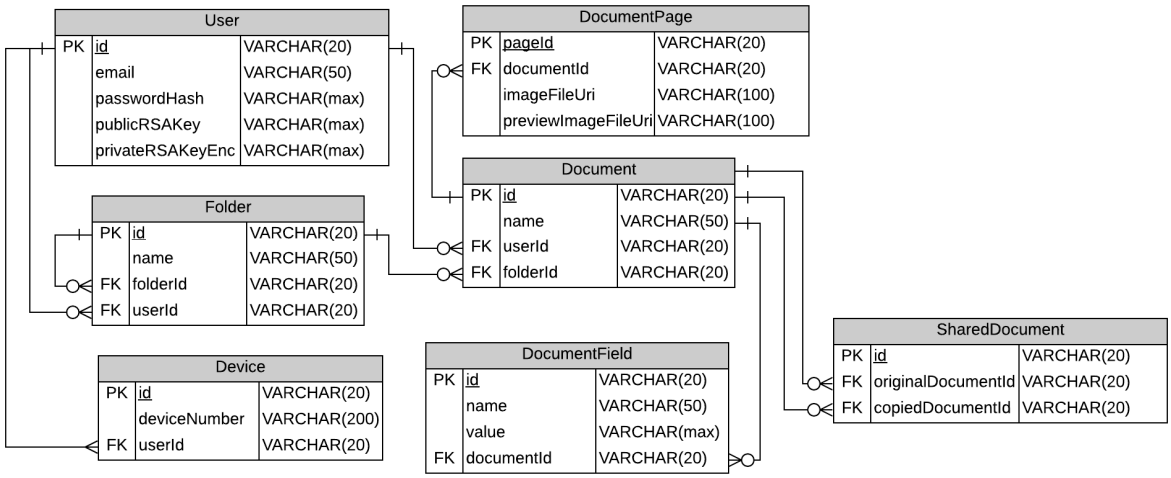


Рис. 3.8. Схема бази даних

Таблиця Folders зберігає інформацію про папки користувачів для групування документів. Вона містить наступні поля:

- id – унікальний ідентифікатор папки;
- name – назва папки;
- folderId – зовнішній ключ, що є ідентифікатором папки у якій розміщується дана папка. Значення може бути NULL, що означає, що дана папка знаходиться в так званій рутовій папці, тобто відображається відразу при вході в застосунок;
- userId – зовнішній ключ, що є ідентифікатором користувача, який є власником папки.

Таблиця Documents зберігає інформацію про документи користувачів. Вона містить наступні поля:

- id – унікальний ідентифікатор документу;
- name – назва документу;
- folderId – зовнішній ключ до таблиці Folders, що є ідентифікатором папки у якій розміщується даний документ. Значення може бути NULL, що означає, що даний документ

знаходиться в так званій рутовій папці, тобто відображається користувачу відразу при вході в застосунок;

- userId – зовнішній ключ, що є ідентифікатором користувача, який є власником документа.

Таблиця DocumentFields зберігає інформацію про дані у документі ,у вигляді ключ – значення в зашифрованому вигляді. Вона містить наступні поля:

- id – унікальний ідентифікатор для пари;
- name – назва поля документу, наприклад для паспорту, користувач може ввести значення: серія, номер, ким виданий і т.д.;
- value – значення поля документу, наприклад для серії паспорта це може бути СК;
- documentId – зовнішній ключ до таблиці Documents, що є ідентифікатором документа в якому заповнювалось поле.

Таблиця DocumentPages використовується для зберігання метаданих для відображення сторінок документу. Вона містить наступні поля:

- pageId – унікальний ідентифікатор для сторінки документа;
- documentId – унікальний ідентифікатор документа, якому належить сторінка;
- imageFileUri – посилання на файл зображення документу;
- previewImageFileUri – посилання на файл зображення документу, адаптованого для попереднього перегляду на клієнтському пристрої.

Таблиця SharedDocuments використовується для управління документами, що користувач поширює для інших користувачів. Вона містить наступні поля:

- id – унікальний ідентифікатор для рядка таблиці;

- originalDocumentId – зовнішній ключ до таблиці Documents, що є ідентифікатором документа, до якого був наданий доступ іншому користувачу;
- copiedDocumentId – зовнішній ключ до таблиці Document, що є ідентифікатором документа, який був створений для поширення іншому користувачеві.

Таблиця Users зберігає інформацію про користувачів застосунку. Вона містить наступні поля:

- id – унікальний ідентифікатор користувача, що присвоюється йому сервісом автентифікації;
- publicRSAKey – публічний ключ RSA користувача, який використовується для шифрування даних для поширення доступу;
- privateRSAKeyEnc – зашифрований приватний ключ RSA користувача, який використовується для розшифрування даних на локальному пристрої;
- email – електронна адреса користувача;
- passwordHash – хеш автентифікації користувача, що використовується для автентифікації запитів на серверній частині.

Таблиця Devices використовується для зберігання інформації про пристрої користувачів для надсилання повідомлень із серверної частини. Вона містить наступні поля:

- id – унікальний ідентифікатор пристрою;
- deviceNumber – номер пристрою, що використовується сервісом відправки повідомлень для ідентифікації пристрою;
- userId – зовнішній ключ, що є ідентифікатором користувача, який є власником пристрою.

Зв’язки між таблицями винесено у табл. 3.1.

Таблиця 3.1

Зв’язки між таблицями у базі даних

№	Сутність	Сутність	Тип зв’язку
1	Folder	Folder	Один до багатьох
2	Folder	Document	Один до багатьох
3	User	Folder	Один до багатьох
4	User	Document	Один до багатьох
5	User	Device	Один до багатьох
6	Document	DocumentField	Один до багатьох
7	Document	DocumentPage	Один до багатьох
8	Document	Document	Багато до багатьох через таблицю SharedDocuments

3.3. Алгоритм для шифрування чутливих даних

При реєстрації користувача у клієнтському застосунку, він створює пароль. Цей пароль використовується для автентифікації в обліковому записі користувача сервісу за допомогою клієнтського застосунку. Віддалене сховище розроблене таким чином, щоб запобігти можливості розшифрування даних, що зберігаються в ньому без використання паролю користувача.

Для цього використано локальну модель шифрування, що забезпечує дозвіл розшифрувати та отримати доступ до своїх даних лише власнику головного ключа. Всі чутливі дані сховища шифруються та розшифровуються виключно на стороні клієнтського застосунку, до того як

він синхронізується з хмарним сервером у зашифрованому вигляді. Хмарні сховища не мають доступ до пароля, а також не зберігають його. Дані, що зберігаються на серверах, розшифровуються, коли це потрібно для використання, на мобільному пристрої користувача, після успішного введення пароля у клієнтському застосунку.

Для шифрування даних використано алгоритми шифрування та хешування найвищого стандарту для захисту даних користувачів. Коли користувач створює свій обліковий запис, створюється хеш пароля з використанням солі користувача. Сіль користувача – це випадковий рядок, що генерується окремо для кожного користувача. Цей крок виконується на пристрої користувача (на стороні клієнта) (див. Додаток 1. Схема процесу шифрування персональних даних).

Для отримання хеша паролю, який є ключем для шифрування даних треба виконати n ітерацій алгоритму PBKDF2-SHA256. З використанням хеша головного паролю виконується ще одна ітерація хешування для генерації хеша автентифікації пароля (або "хеша для входу"). Хеш автентифікації надсилається на сервер і проходить ще n ітерацій хешування PBKDF2, для генерації ключа автентифікації, який зберігається на сервері, для виконання автентифікації при обробці подій, надісланих з пристрою користувача для зміни хмарної бази даних. Коли Lambda обробник отримує запит з черги подій, хеш порівнюється з ключем автентифікації користувача, що зберігається в базі даних сховища.

Таким чином пароль та ключ шифрування ніколи не надсилаються на сервер. А оскільки хешування – це односторонній алгоритм, сервер не може змінити хеш автентифікації, який він отримує.

3.4. Алгоритм для поширення доступу на зашифровані дані

Для того, щоб дозволити користувачам ділитися даними облікових записів із синхронізованими сторонами використовується алгоритм RSA.

Користувачі можуть створювати спільні документи, щоб надати відповідний доступ для іншого користувача.

RSA використовує асиметричні алгоритми шифрування, де ключ, який використовується для шифрування повідомлення, відрізняється від ключа, що використовується для його розшифрування. У кожного користувача є пара криптографічних ключів, один загальнодоступний, один приватний. Відкритим ключем можна поділитися з будь-яким іншим користувачем і він використовується для шифрування даних, тоді як приватний ключ доступний лише для користувача і використовується для розшифрування даних, зашифрованих за допомогою відповідного відкритого ключа (рис. 3.9).

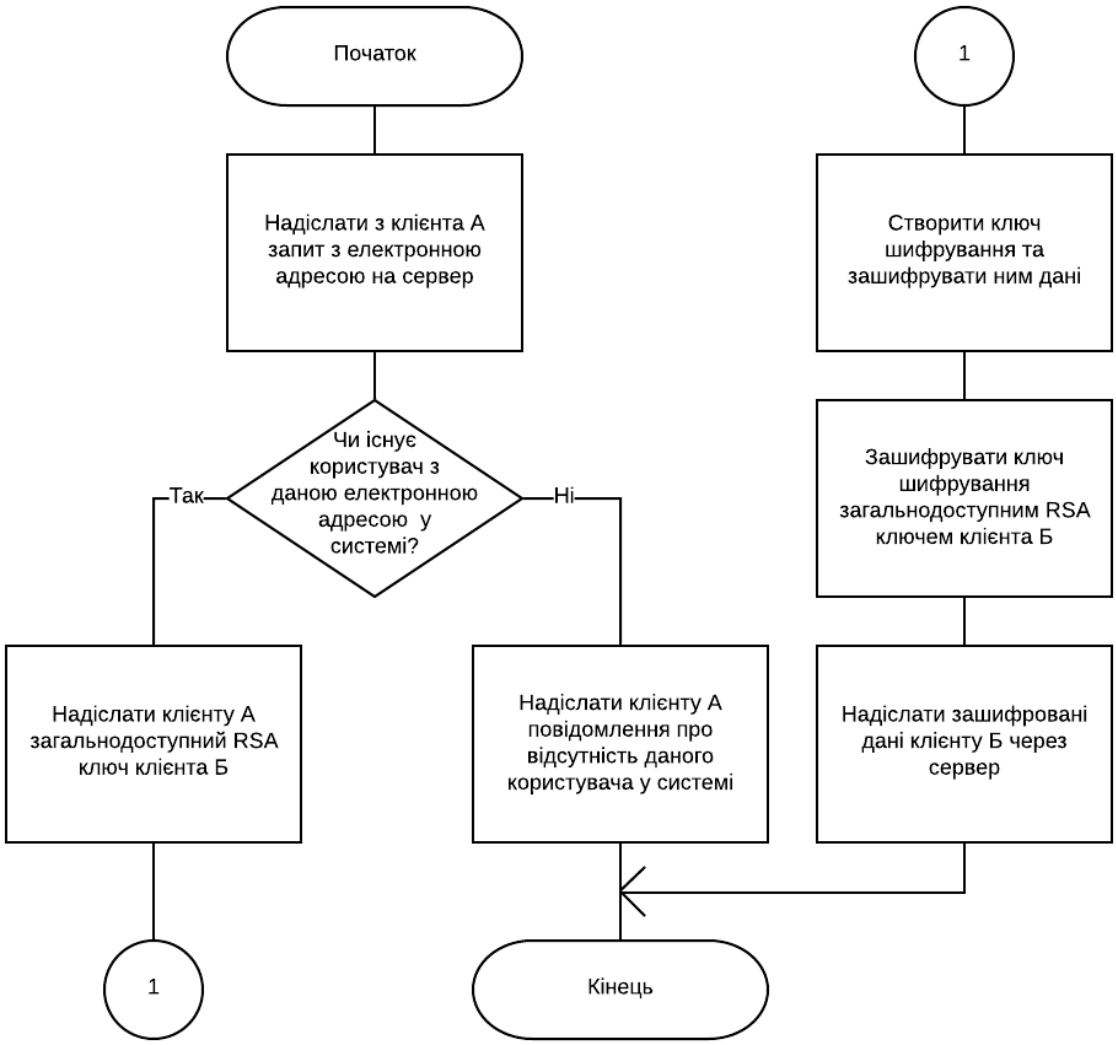


Рис. 3.9. Схема алгоритму шифрування даних для поширення

При створенні “спільного документу” створюється 256-бітний ключ шифрування. Також при реєстрації користувача генерується 2048-бітна пара ключів RSA локально на клієнтському пристрої. Далі приватний ключ користувача зашифровується ключем шифрування сховища, використовуючи AES-256-бітне шифрування, після чого надсилається на сервер разом із відкритим ключем користувача. Зашифрований приватний ключ надсилається в хмарне сховище, щоб його можна було отримати з інших пристроїв у майбутньому. Відкриті ключі будуть використовуватися іншими користувачами для шифрування даних, які можна розшифрувати лише за допомогою оригінального приватного ключа.

4. АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Особливості реалізації розробленої системи

4.1.1. Особливості реалізації серверної частини

Серверна частина у вигляді Lambda обробника реалізована з використанням середовища виконання .NET Core версії 3.1 мовою програмування C#.

Для взаємодії обробника з базою даних використовується бібліотека Entity Framework (EF) Core. Дана бібліотека являє собою об’єктно-орієнтовану, розширювану технологію для доступу до даних. EF Core є ORM-інструментом, тобто дозволяє працювати з базами даних, але надає розробнику більш високий рівень абстракції [27]. Таким чином, EF Core дозволяє розробнику абстрагуватися від самої бази даних та її таблиць і працювати з даними, незалежно від типу сховища, в якому вони зберігаються. На фізичному рівні розробник повинен оперувати таблицями, індексами, первинними та зовнішніми ключами, в той час на рівні абстракції, що пропонує Entity Framework, він має працювати з об'єктами.

Основна концепція Entity Framework включає поняття сутності. Саме сутність визначає набір даних, які пов’язані з визначеним об'єктом. Через це, дана технологія пропонує роботу не з таблицями, а з об'єктами та їх колекціями.

Сутності, що описані для серверної частини:

- Folder;
- Document;
- User;
- DocumentField;
- DocumentPage;
- SharedDocument.

Для вибірки даних з БД використовується LINQ (Language Integrated Query). З використанням LINQ можна створювати різні

запити для зручного вибору об’єктів з бази даних, в тому числі, які можуть бути пов’язані різними асоціативними відношеннями. Entity Framework транслює запити LINQ у SQL вирази, які є зрозумілими для окремо взятої СУБД.

4.1.2. Особливості реалізації взаємодії клієнтської та серверної частини

Для реалізації відправки повідомлень з клієнтської частини в чергу повідомлень використовується пакет “aws_sqs_api” для фреймворку Flutter. Він дозволяє здійснювати взаємодію з сервісом AWS SQS через API використовуючи протокол HTTPS. Для виклику API сервісу, клієнту необхідні наступні параметри для формування хедеру авторизації HTTP запиту: access key ID та secret access key. Таким чином, сервіс точно ідентифікує, який клієнт надіслав повідомлення. Це дозволяє захистити чергу від небажаних повідомлень наступним чином:

- перевірка сутності, що відправляє запит: підпис гарантує, що запит був відправлений кимось, у кого є дійсний ключ доступу;
- захист даних під час передачі: для того, щоб запобігти підробці запиту під час його передачі, деякі елементи запиту використовуються для обрахування хешу запиту, та відповідне значення хеш-функції включається як частина запиту. Коли сервіс AWS SQS отримує запит, він використовує ту ж саму інформацію для обрахування хешу та співставляє її зі значенням хешу в запиті. Якщо ці значення не співпадають, то запит відхиляється;
- захист від можливих атак повторного відтворення: у більшості випадків запит має досягти AWS протягом п’яти хвилин після відмітки часу в запиті. В іншому випадку AWS відхилить запит.

4.1.3. Особливості реалізації клієнтської частини

Для реалізації клієнтської частини програмної системи було використано крос-платформний фреймворк для розробки мобільних застосунків – Flutter. Його основна ідея полягає в тому, що інтерфейс створюється у вигляді, так званого, дерева віджетів. Віджети описують те, як повинен виглядати екран, враховуючи поточну конфігурацію та стан. Коли стан віджета змінюється, він швидко оновлює свій вигляд, завдяки тому, що фреймворк порівнює попередній стан віджету з оновленим та аналізує, які мінімальні зміни необхідні в дереві візуалізації для переходу з одного стану в інший та рендеру екрану.

Далі наведено список основних класів програмного коду, що описують екрани застосунку:

- SplashScreen – описує екран-заставку, що з’являється під час завантаження програми;
- LoginScreen – описує екран, що відображається при автентифікації користувача в застосунку;
- RegisterScreen – описує екран, що відображається при реєстрації користувача в застосунку;
- MainScreen – описує основний екран, що відображає список папок та файлів, які містяться в обраній папці;
- DocumentDetailsScreen – описує екран, що відображає деталі для вибраного документу, та дозволяє здійснювати операції над ним;
- AddEditDocumentScreen – описує екран, що містить форму для додавання та оновлення документу;
- AddEditFolderScreen – описує екран, що містить форму для додавання та оновлення папки;
- DocumentPreviewScreen – описує екран, що містить зображення зі сторінками документу для їх перегляду.

Для створення зображень документів використовується бібліотека Scanbot SDK. Дана бібліотека працює повністю в автономному режимі. Це

означає, що для її роботи не потрібно вмикати Інтернет з'єднання на клієнтському пристрої. Усі дані, що генеруються за допомогою SDK Scanbot, зберігаються лише на пристрої кінцевих користувачів і ні в якому разі ніколи не передаються на інший сервер / хмарний сервіс.

Основна функціональність бібліотеки Scanbot SDK, що використовується в застосунку:

- автоматичне виявлення документу;
- кадрування зображення документу;
- створення файлу у форматі PDF.

4.1.4. Особливості реалізації сервісу автозаповнення

Для реалізації сервісу автозаповнення (див. пункт 3.1.7) було розроблено окремий модуль, який обробляє метадані форми та повертає результуючий сет даних, які користувач може використати для автозаповнення. Цей модуль взаємодіє з локальною базою даних, щоб отримати інформацію, що зберігають документи та модулем шифрування, щоб розшифрувати значення полів документів для заповнення, оскільки ці дані зберігаються в зашифрованому вигляді (рис. 4.1).

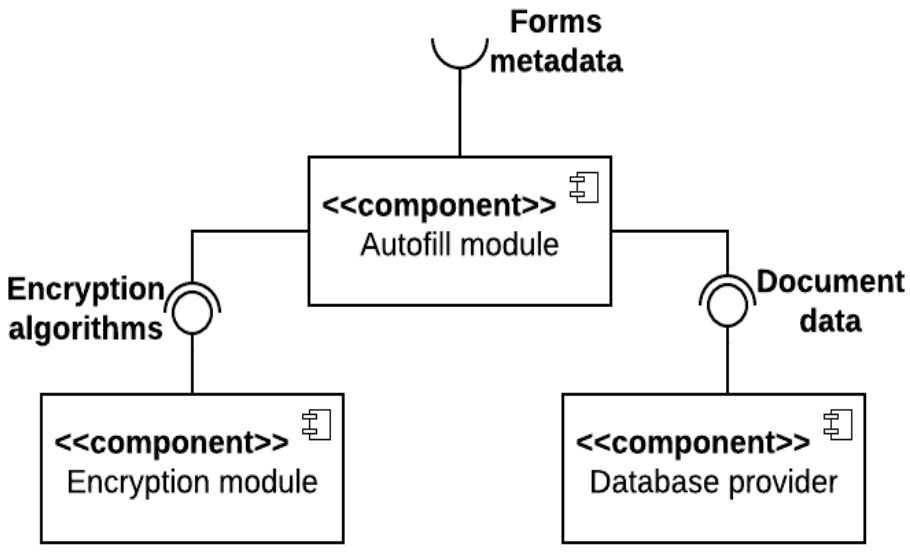


Рис. 4.1. Діаграма компонент сервісу автозаповнення

Метадані форми – це дані, на основі яких буде виконуватись пошук та власне та інформація, яка дозволяє ідентифікувати, які саме значення необхідно надати сервісу для автозаповнення. Наприклад, форми веб-сайтів містять елементи <input> для введення даних. Ці елементи мають атрибут “name”, який має містити логічну назву, для відповідного поля форми. Для мобільних застосунків операційної системи Android також існують спеціальні атрибути для елементів форми, що дозволяють надавати “підказки” для сервісу автозаповнення (лістинг 4.1).

Лістинг 4.1. Приклад елементу форми мобільного застосунку

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:autofillHints="password" />
```

На основі значення цього атрибуту здійснюється пошук по таблиці DocumentFields, а саме: співставляються на схожість значення поля name даної таблиці локальної бази даних та значення надані у метаданих форми.

4.2. Дизайн та вміст екранів

При запуску мобільного застосунку, для неавтентифікованого користувача відкривається екран входу. На ньому відображається форма для автентифікації та кнопки, за допомогою яких можна перейти на екран реєстрації, або увійти в застосунок. Кнопка “Увійти” є неактивною, доки користувач не заповнить поля відповідними валідними даними для автентифікації: електронну адресу та пароль користувача. Ці поля валідуються за допомогою регулярних виразів. На рис. 4.2 та рис. 4.3 зображена форма входу, що заповнена валідними та невалідними значеннями для полів вводу.

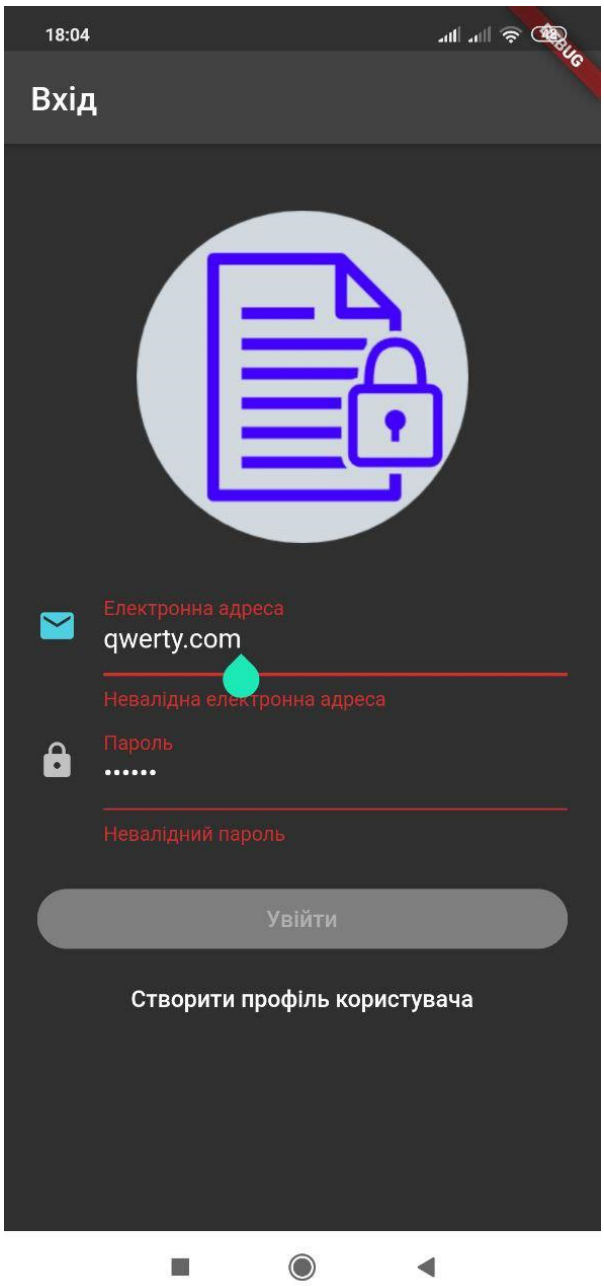


Рис. 4.2. Неуспішна валідація

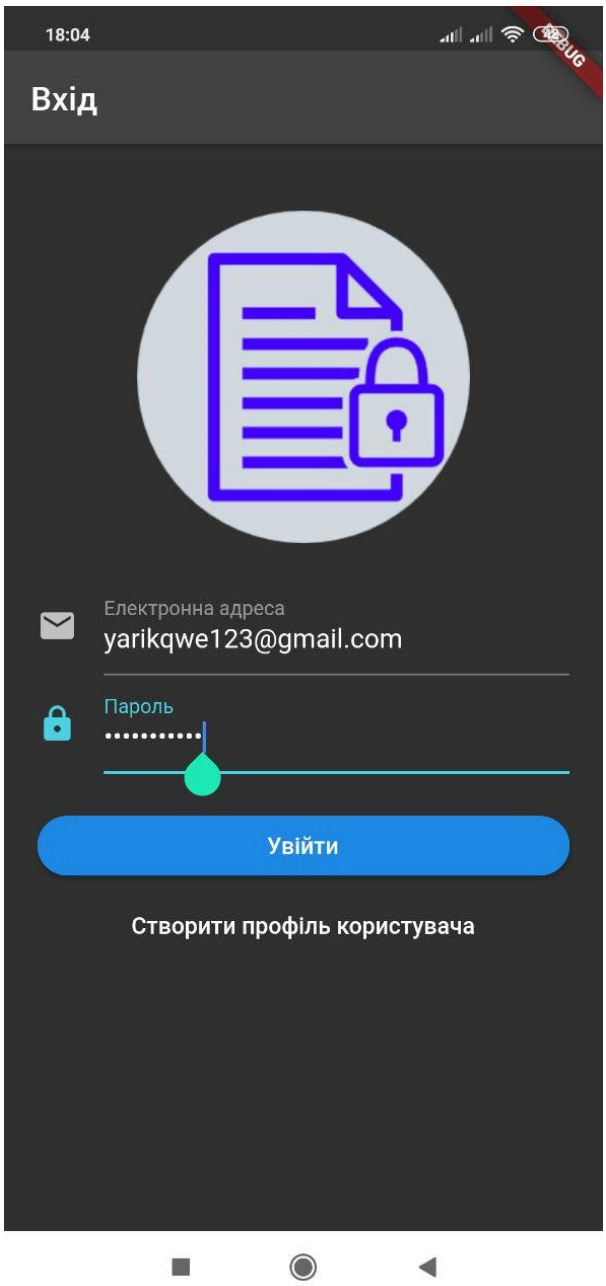


Рис. 4.3. Успішна валідація

Якщо користувач не має існуючого профілю, то йому необхідно зареєструватися у застосунку. Після переходу на екран реєстрації, відображається форма реєстрації. Вона містить поля для вводу електронної адреси та паролю користувача. Також ці поля валідуються аналогічним способом, як і для форми автентифікації. Кнопка “Реєстрація” є неактивною, доки значення полів не стануть валідними. На рис. 4.4 та рис. 4.5 зображена форма реєстрації, що заповнена валідними та невалідними значеннями для полів вводу.

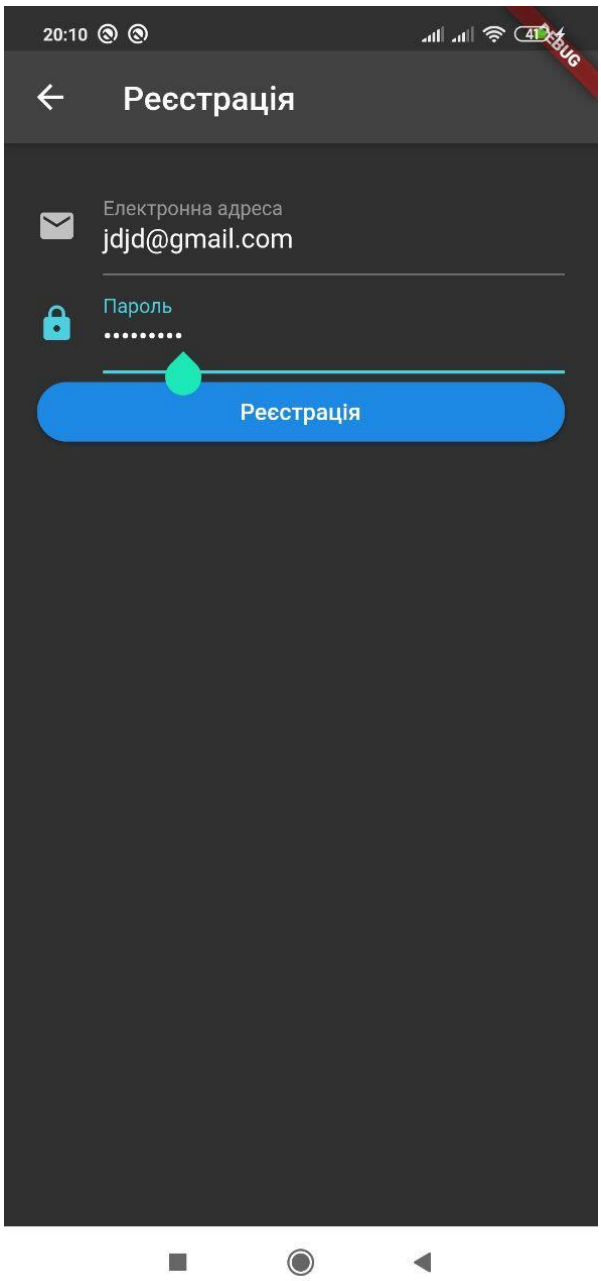


Рис. 4.4. Успішна валідація

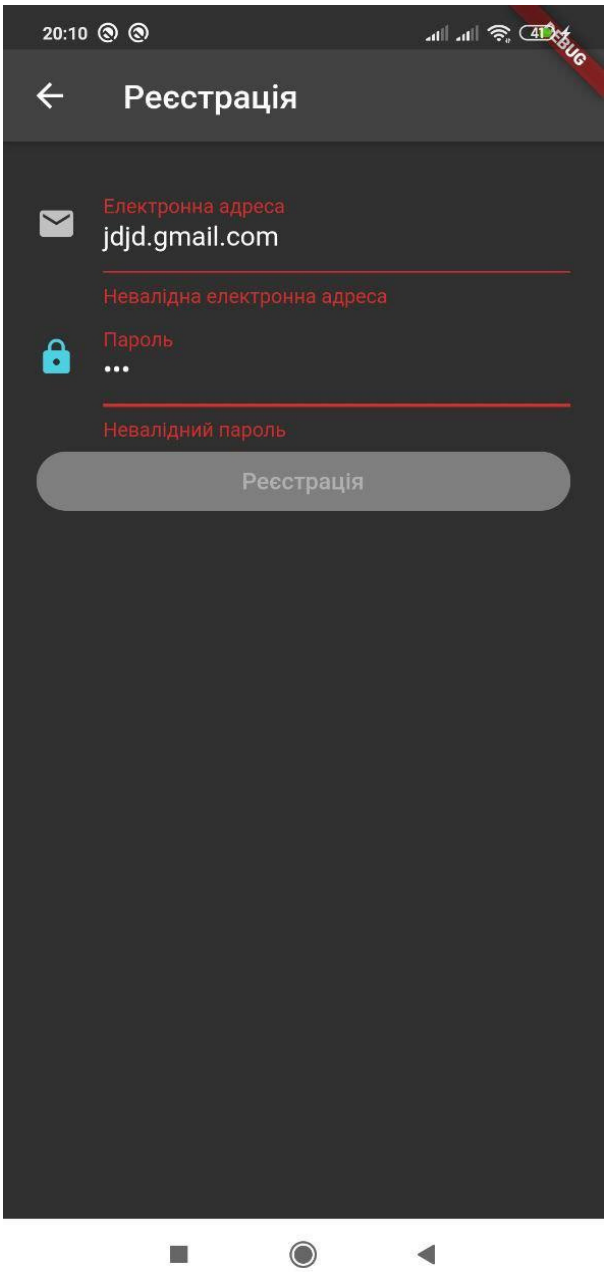


Рис. 4.5. Неуспішна валідація

Після успішної реєстрації або автентифікації, відкривається головний екран, що відображає список файлів та папок користувача. Головний екран має навігацію у вигляді табів, що розташовуються на нижній панелі екрану. У табі з назвою “Мої” відображаються документи та папки, власником яких є авторизований користувач (рис. 4.6). У табі “Спільні” відображаються документи, якими інші користувачі поділились з авторизованим (рис. 4.7). Активна таба підсвічується додатковим кольором. Якщо таба “Мої” обрана активною, то над панеллю управління табами доступна кнопка з

символом “+” для додавання нової папки, або нового документа. При її натисканні замість панелі табів з’являється додаткове меню для вибору вікна створення: папки або документу.

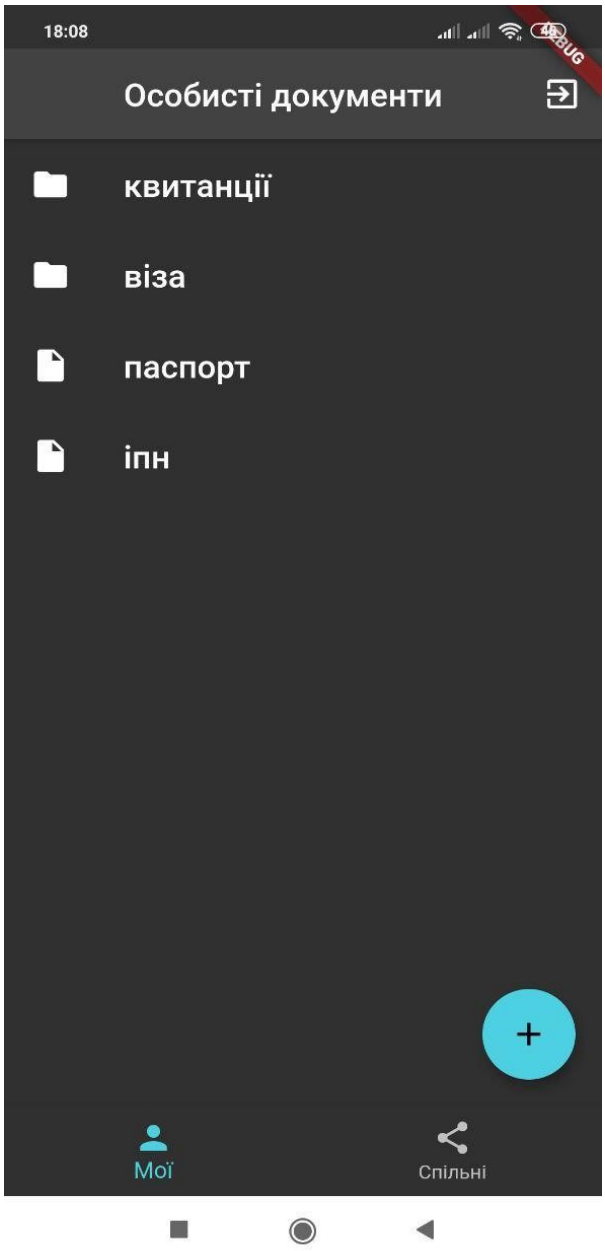


Рис. 4.6. Активна таба “Мої”

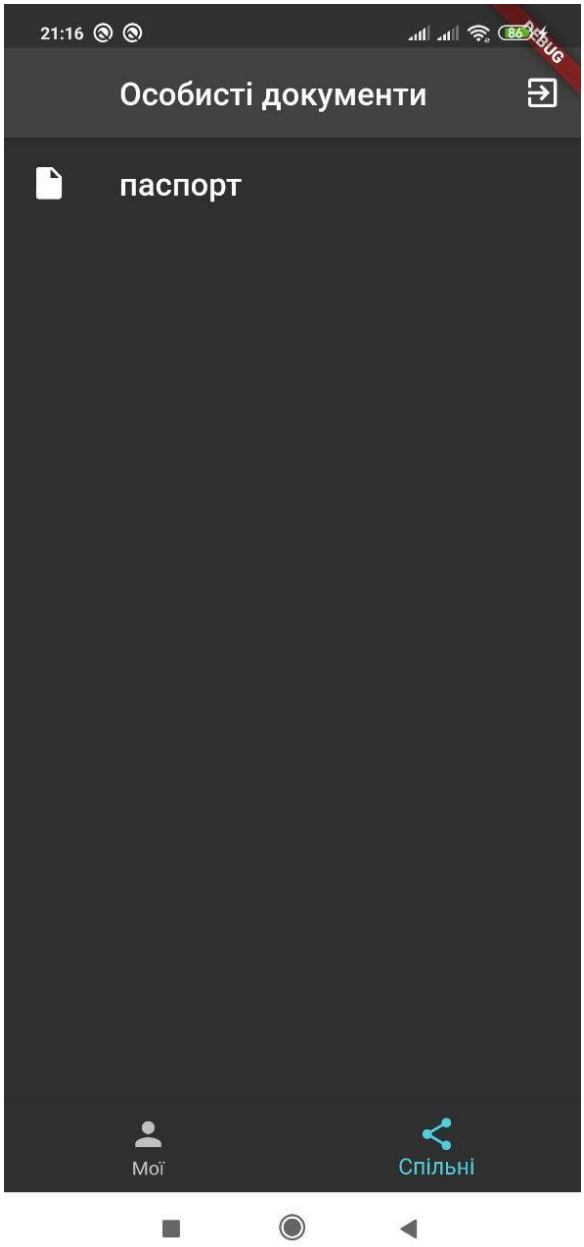


Рис. 4.7. Активна таба “Спільні”

Верхня панель головного екрану відображає назву поточної папки та має кнопку виходу з поточного профілю. У випадку, якщо це, так звана, рутова папка, то панель містить текст “Особисті документи”.

Екран для додавання та редагування папки має форму, що містить поле для значення назви папки (рис. 4.8). Екран для додавання та

редагування документу має форму, що містить поле для значення назви документу та динамічні поля для додавання будь-якої кількості довільних пар ключ-значення, що містять дані документу (рис. 4.9).

Рис. 4.8. Створення папки

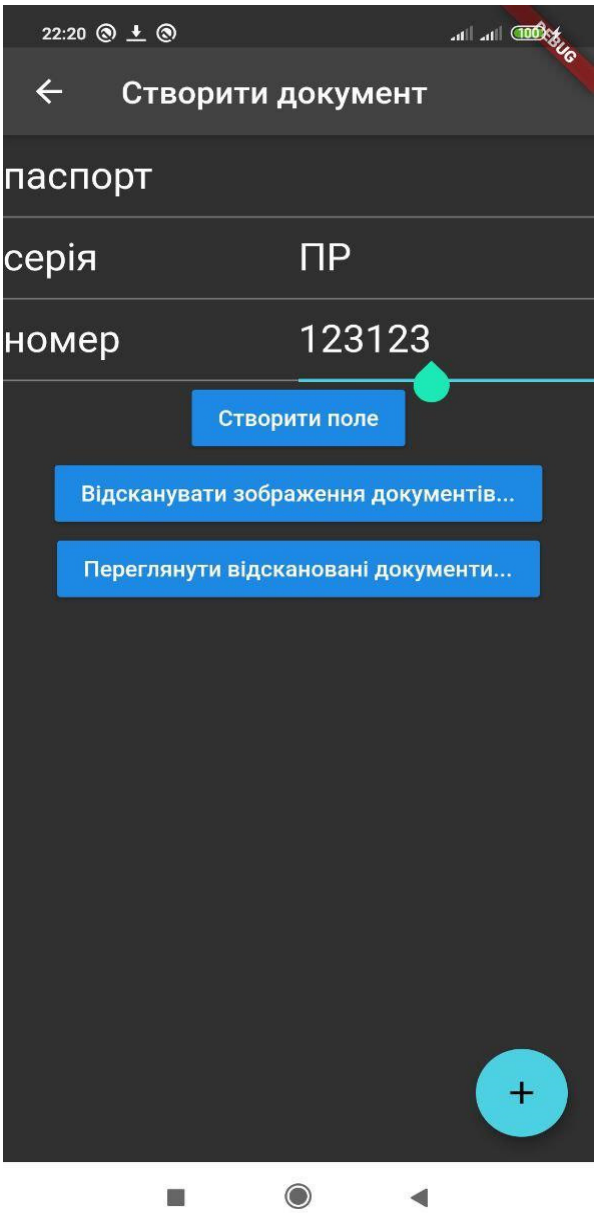


Рис. 4.9. Створення документу

Також екран документу містить кнопки для переходу на екрани сканування (рис. 4.10) та попереднього перегляду зображень документу (рис. 4.11). Екран сканування відображає поточне зображення з камери мобільного пристрою та зеленим прямокутником автоматично виділяє область, що була ідентифікована як документ. На екрані також

з’являються підказки, якщо кут нахилення пристрою не дозволяє зробити чіткий знімок. Екран перегляду зображень показує поточні зображення документу у зменшеному форматі.

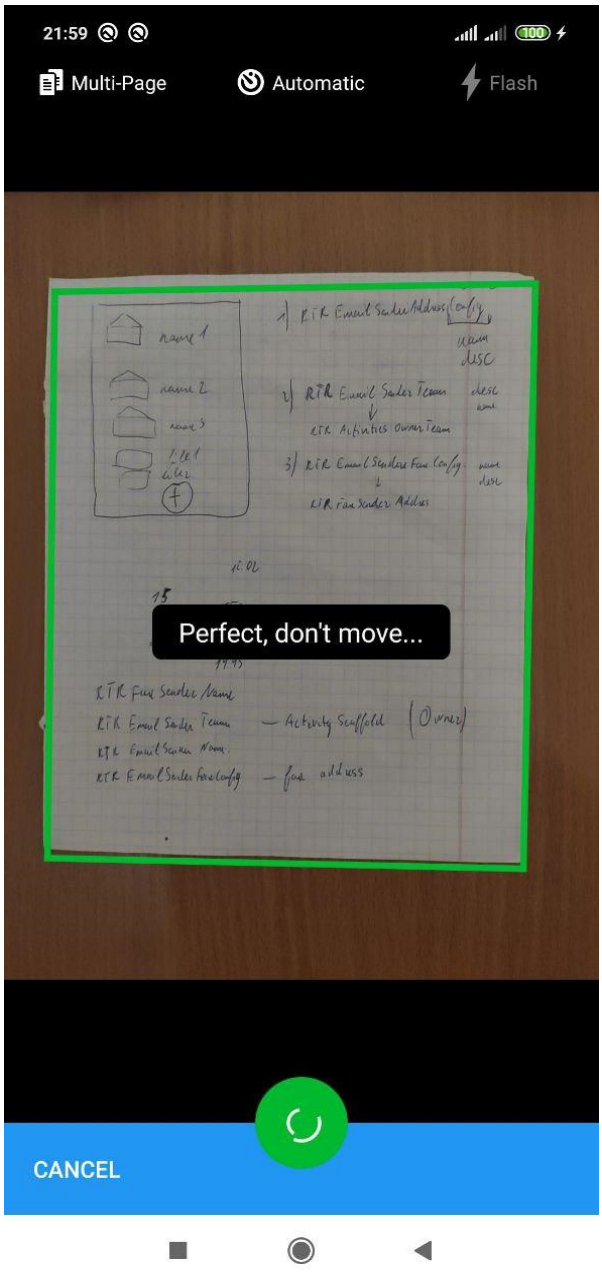


Рис. 4.10. Сторінка сканування

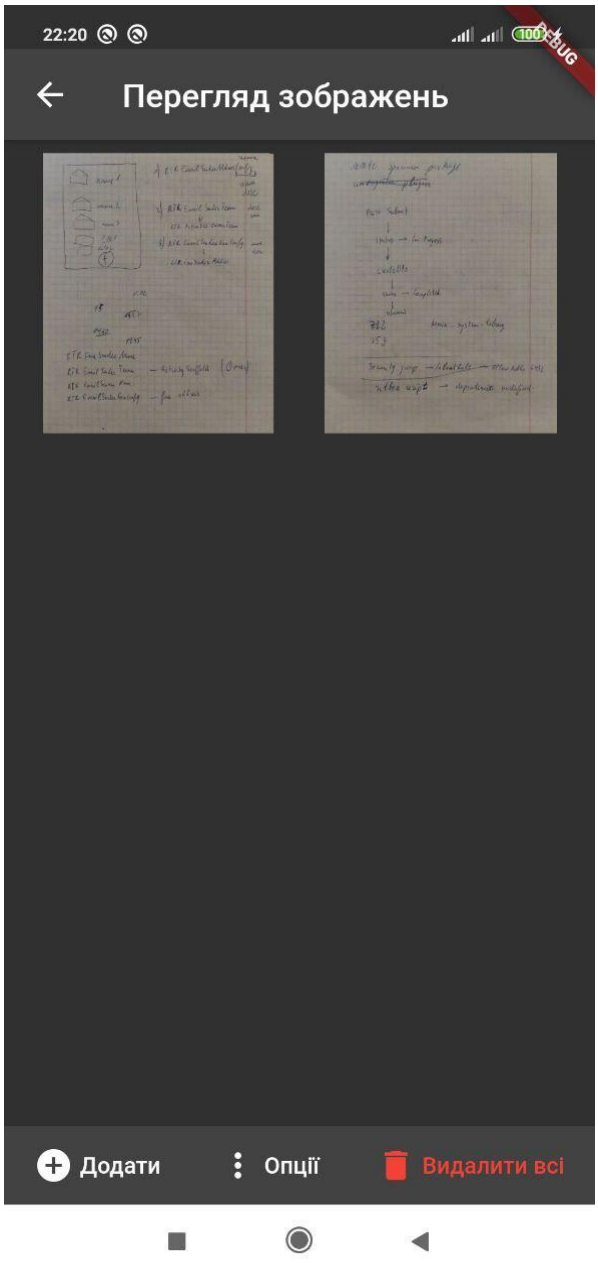


Рис. 4.11. Сторінка перегляду

Для переходу в іншу папку необхідно натиснути на неї в списку головного меню. Після цього в головне меню завантажується список файлів та папок, що знаходяться всередині обраної. Верхня панель відображає назву папки, в якій зараз знаходиться користувач. Для того, щоб редагувати,

або видалити папку необхідно натиснути і потримати кілька секунд. Після цього з’явиться меню з опціями “Редагувати” та “Видалити” папку. Якщо натиснути кнопку “Редагувати”, то відкриється екран з формою для редагування назви папки. Якщо натиснути кнопку “Видалити”, то папка видаляється, разом з усіма іншими наявними всередині неї документами та папками.

Для того, щоб перейти на екран детального перегляду документу необхідно натиснути на нього у списку головного меню. Кнопка зі знаком “+”, що розташована внизу екрану, дозволяє перейти на екран редагування документу. Також, на екрані детального перегляду у верхній панелі стає доступне меню з наступними опціями: “Видалити” та “Поділитися”. При натисканні кнопки “Видалити” – документ видаляється. Якщо натиснути кнопку “Поділитися”, з’явиться меню з полем вводу для електронної адреси користувача, з яким необхідно поділитися даним документом. Користувач, з введеною електронною адресою матиме у списку “Спільні” даний документ.

4.3. Тестування системи

Для проведення якісного димового (smoke) тестування, було сформовано набір тестових випадків (тест-кейсів), відповідно до вимог до розроблюваної системи. Після цього було проведено тестування на відповідність поставленим у технічному завданні вимогам. Для тестування використано мобільний пристрій з встановленою ОС Android версії 10 QKQ1.190828.002. Результати попереднього тест-кейсу можуть використовуватися у подальших тест-кейсах, оскільки усі тести перевіряються у порядку їх опису нижче (табл. 4.1).

Було складено наступні сценарії перевірки:

1. Зареєструватись в системі.
2. Авторизуватись в системі з використанням існуючого профілю користувача.

3. Створити нову папку.
4. Редагувати папку.
5. Видалити папку.
6. Створити новий документ.
7. Редагувати документ.
8. Видалити документ.
9. Поділитися документом з іншим користувачем.
10. Використати автозаповнення для форми іншого застосунку.

Таблиця 4.1

Опис тестових випадків

№	Опис дій	Очікуваний результат
1	1. Запустити застосунок. 2. Натиснути кнопку “Створити профіль користувача”. 3. Ввести валідні електронну адресу та пароль. 4. Натиснути кнопку “Реєстрація”.	Користувач зареєстрований у системі. Відкривається основний екран з порожнім списком документів та папок.
2	1. Запустити застосунок. 2. Ввести електронну адресу та пароль. 3. Натиснути кнопку “Увійти”.	У разі успішної авторизації відкривається основний екран зі списком документів та папок користувача.
3	1. Відкрити табу “Мої”. 2. Натиснути кнопку зі знаком “+”. 3. Обрати з меню “Папку”. 4. Ввести назву папки у формі на екрані створення. 5. Натиснути кнопку зі знаком “+”.	Створена папка відображається в списку відповідної папки.

Продовження табл. 4.1

№	Опис дій	Очікуваний результат
4	1. Натискати на папку кілька секунд. 2. Обрати з меню, що з’явилося, опцію “Редагувати”. 3. Змінити назву папки у формі на екрані редагування. 4. Натиснути кнопку зі знаком “✓”.	Обрана папка відображається у списку відповідної папки з оновленою назвою.
5	1. Натискати на папку кілька секунд. 2. Обрати з меню, що з’явилося, опцію “Видалити”.	Обрана папка разом з її вмістом видалена зі списку відповідної папки.
6	1. Відкрити табу “Мої”. 2. Натиснути кнопку зі знаком “+”. 3. Обрати з меню “Документ”. 4. Ввести назву документу. 5. Додати поля документа. 6. Відсканувати зображення для документа. 7. Натиснути кнопку зі знаком “+”.	Створений документ відображається в списку відповідної папки.

Продовження табл. 4.1

№	Опис дій	Очікуваний результат
7	1. Відкрити екран перегляду документу, натиснувши на документ. 2. Натиснути кнопку в нижній частині екрану. 3. Змінити назву документу. 4. Змінити назву одного з полів документу. 5. Змінити значення одного з полів документу. 6. Додати нове зображення для документу. 7. Натиснути кнопку зі знаком “✓”.	Обраний документ відображається у списку відповідної папки з оновленою назвою. Детальна інформація про документ також оновлена.
8	1. Відкрити екран перегляду документу, натиснувши на документ. 2. Вибрати опцію “Видалити” з додаткового меню у верхній панелі екрану.	Документ видалено зі списку відповідної папки.
9	1. Відкрити екран перегляду документу, натиснувши на документ. 2. Вибрати опцію “Поділитися” з додаткового меню у верхній панелі екрану. 3. Ввести електронну адресу користувача у поле на додатковій формі, що з’явилася на екрані. 4. Натиснути кнопку “Ок”.	Обраний документ з’являється у списку “Спільні” користувача електронна адреса якого була введена в поле на додатковій формі.

Продовження табл. 4.1

№	Опис дій	Очікуваний результат
10	1. Відкрити форму іншого застосунку. 2. Навести фокус на поле форми для заповнення. 3. Обрати один із запропонованих сетів даних для заповнення.	При наведенні фокусу на поле форми, з’являється один, або кілька варіантів для заповнення поля. Поле форми заповнене даними.

4.4. Рекомендації, щодо подальшого вдосконалення

Компоненти розробленого програмного забезпечення можуть бути удосконаленими. Проаналізувавши реалізовані компоненти ПЗ після тестування, було виділено наступні рекомендації щодо вдосконалення даного рішення:

1. Розширити процес автентифікації та реєстрації, шляхом додавання більше функціональності для взаємодії з профілем користувача. Наприклад, надати можливість зміни паролю, додати можливість підтвердження електронної адреси, задля запобігання введення неіснуючих електронних адрес.
2. Замість повного шифрування бінарних файлів скан-копій документів можна використати часткове шифрування файлів. Це дозволить збільшити продуктивність застосунку та зменшити час на шифрування. Оскільки навіть частково зашифрований файл майже неможливо повністю відтворити у випадку витоку даних.
3. Покращити процес створення документів, шляхом застосування технології оптичного розпізнавання символів. Це дозволить автоматично виявляти пари ключ-значення даних на зображеннях документів та значно полегшити роботу користувачу.

ВИСНОВКИ

Метою даного дипломного проєкту було розроблення програмного забезпечення для зберігання та використання персональних документів.

Перед початком розроблення було проведено аналіз моделей зберігання персональних даних та обрано гібридну модель. Дослідження функціональності існуючих аналогів призвело до формулювання вимог до розроблюваного програмного забезпечення, що поєднує у собі найкращі підходи для вирішення проблеми зберігання та використання персональних даних. Проведений аналіз доступних технологій розроблення, мов програмування та інших засобів виявив обґрунтовану доцільність використання хмарної технології AWS Lambda та мови програмування C# для реалізації серверної частини та створення мобільного застосунку, за допомогою фреймворку Flutter, в якості клієнтської частини. В процесі проєктування системи було розроблено та описано алгоритми для шифрування даних.

Розроблена програмна система:

- надає можливість здійснювати основні операції над документами, такі як створення, оновлення, видалення;
- дозволяє ділитися доступом до власних документів іншим користувачам системи;
- забезпечує надійний захист та конфіденційність персональних даних користувачів;
- надає можливість автозаповнення форм персональними даними.

Програмне забезпечення реалізоване у повному обсязі, усі функціональні та нефункціональні вимоги є виконаними, тестування продукту відтворено відповідно до затвердженої програми та методики тестування.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Згуровський, Я.Ю. Аналіз моделей зберігання персональних даних користувачів програмних застосунків [Текст] / Р.А. Гадиняк, Я.Ю. Згуровський // Прикладна математика та комп'ютинг. ПМК, 2019 : дванадцята наук. конф. магістрантів та аспірантів, Київ, 13 – 15 лист. 2019 р. : зб. тез доп. / [редкол.: Дичка І. А. та ін.]. – К. : Просвіта, 2019. – С. 16-21.
2. Dropbox cloud storage review [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.techradar.com/reviews/dropbox-cloud-storage-review.
3. Sync.com Review [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.bitcatcha.com/cloud-storage-reviews/sync-com/.
4. Tresorit Review [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.cloudwards.net/review/tresorit/.
5. LastPass Form Autofill [Електронний ресурс] – Режим доступу до ресурсу: https://www.lastpass.com/autofill.
6. FaaS (Function-as-a-Service) [Електронний ресурс] // Национальная библиотека им. Н. Э. Баумана Bauman National Library. – 2016. – Режим доступу до ресурсу: https://ru.bmstu.wiki/FaaS_(Function-as-a-Service).
7. IaaS (Infrastructure-as-a-Service) [Електронний ресурс] // Национальная библиотека им. Н. Э. Баумана Bauman National Library. – 2017. – Режим доступу до ресурсу: https://ru.bmstu.wiki/IaaS_(Infrastructure-as-a-Service).
8. AWS Lambda vs EC2: Comparison of AWS Compute Resources [Електронний ресурс] – Режим доступу до ресурсу: https://www.simform.com/aws-lambda-vs-ec2/.
9. PostgreSQL: About [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/.
10. MySQL [Електронний ресурс] // Вікіпедія — вільна енциклопедія – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MySQL.

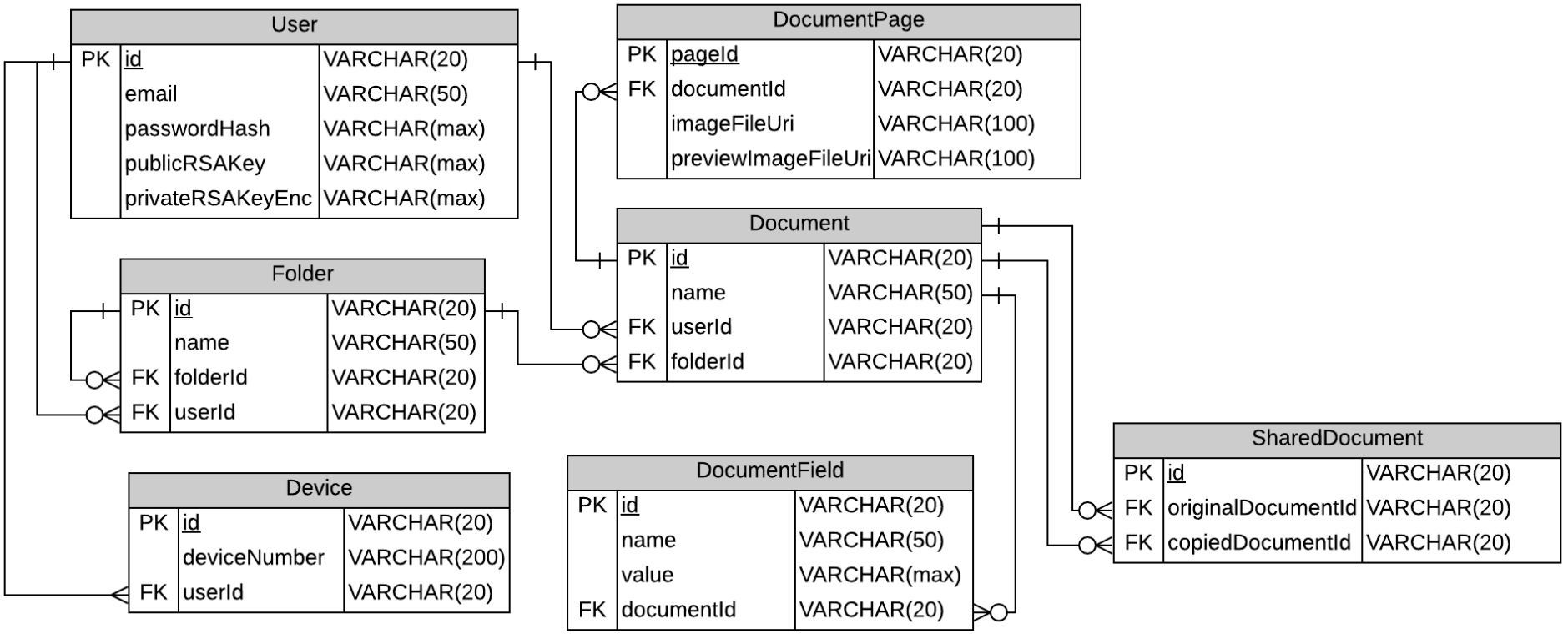
11. Pros and Cons of MySQL 2020 [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.trustradius.com/products/mysql/reviews?qs=pros-and-cons.
12. Round 19 results - TechEmpower Framework Benchmarks [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.techempower.com/benchmarks/#hw=ph&test=plaintext.
13. Выбор между .NET Core и .NET Framework для серверных приложений [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/dotnet/standard/choosing-core-framework-server.
14. Which AWS Lambda programming language should you use? [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://dashbird.io/blog/most-effictient-lambda-language/.
15. Что такое WebExtensions? [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions.
16. 29+ Smartphone Usage Statistics: Around the World in 2020 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://leftronic.com/smartphone-usage-statistics/.
17. Pros and Cons of Mobile Websites and Mobile Apps [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://rubygarage.org/blog/mobile-app-vs-mobile-website.
18. Flutter vs. React Native – What to Choose in 2020? [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2020.
19. Dicjak V. Стоит ли инвестировать во Flutter. Сравнение Flutter и React Native [Електронний ресурс] / Vasiliy Dicjak. – 2019. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/flutter-for-mobile-apps/.

20. Firebase Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/auth.
21. Amazon Simple Queue Service Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/sqs/.
22. What is AWS Lambda? [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.
23. FCM Architectural Overview [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/cloud-messaging/fcm-architecture?hl=ru.
24. Introduction to Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/en_us/AmazonS3/latest/dev/Introduction.html.
25. Autofill framework [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/topics/text/autofill.
26. Matrenin A. Обзор архитектур управления состоянием на Flutter [Електронний ресурс] / Anton Matrenin. – 2019. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/flutter-architecture/.
27. Введение в Entity Framework Core [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://metanit.com/sharp/entityframeworkcore/1.1.php.

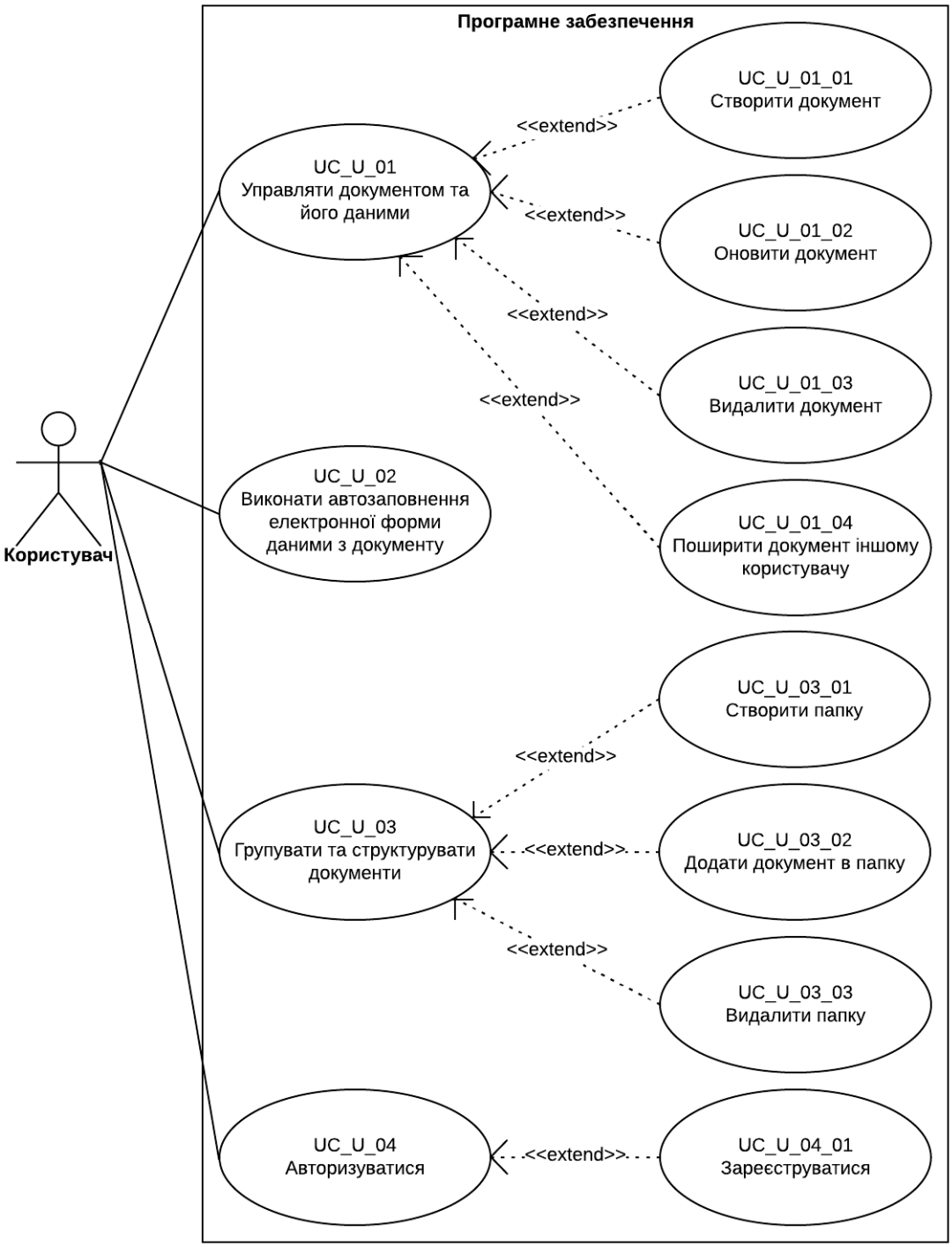
ДОДАТКИ

Додаток 1

Копії графічних матеріалів

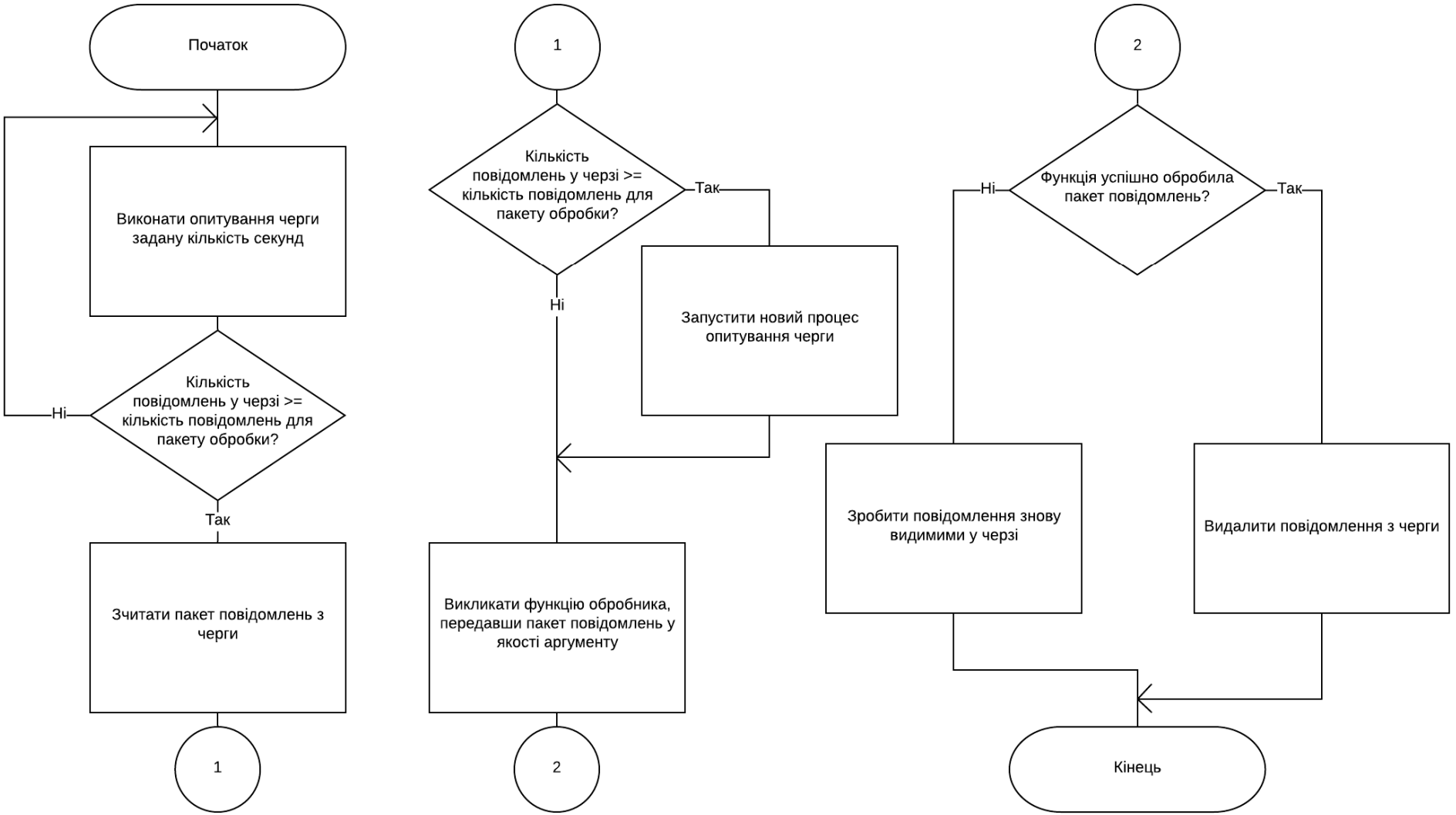


ДП.045440-06-99. Програмне забезпечення для зберігання та використання персональних документів. Структура бази даних. ER-діаграма



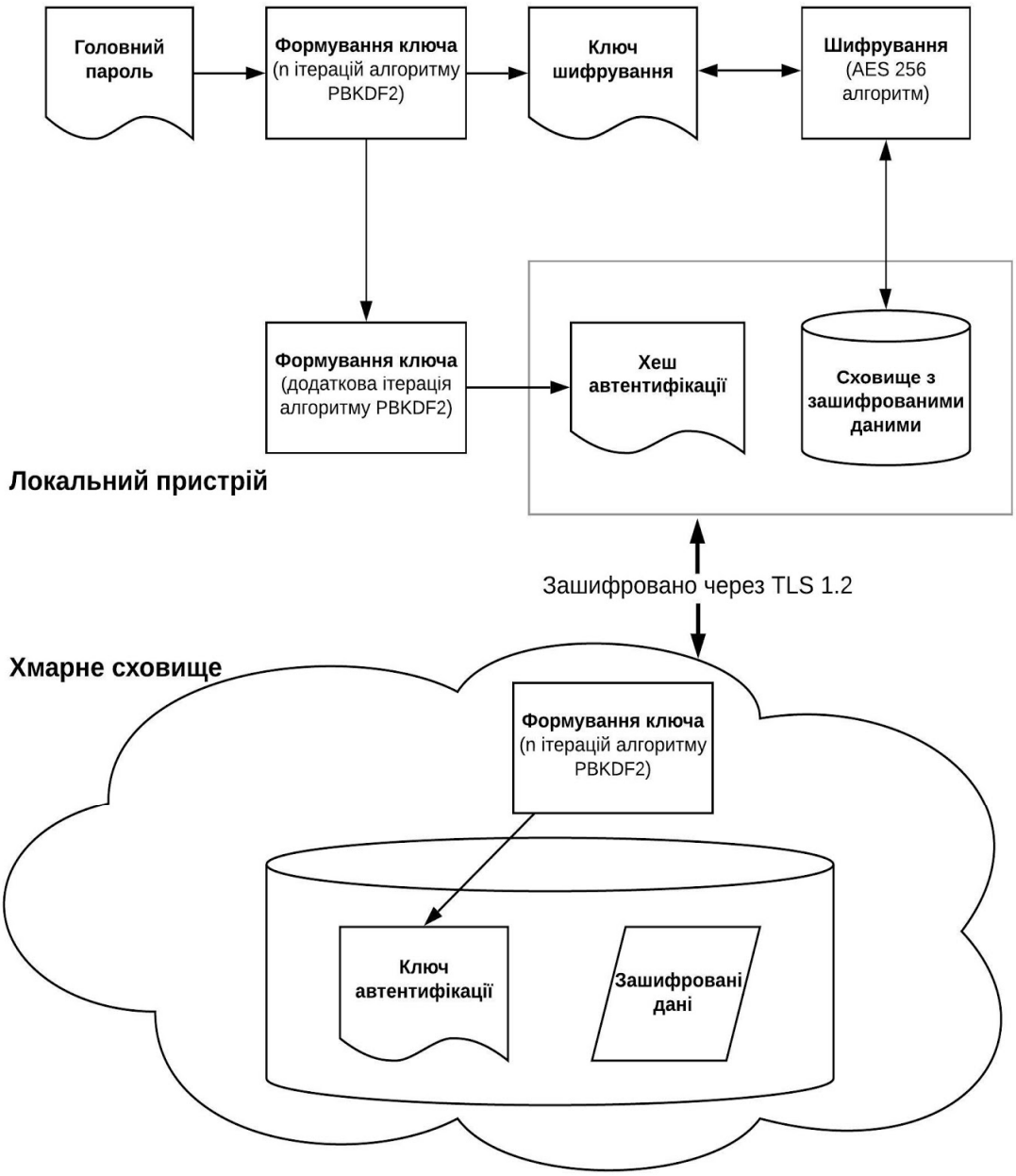
ДП.045440-07-99. Програмне забезпечення для зберігання та використання персональних документів. Функціональність програмних засобів. UML-діаграма прецедентів

Блок-схема алгоритму обробки повідомлень із черги серверної частини



Згуровський Я.Ю., група КП-61

Схема процесу шифрування персональних даних



Згуровський Я.Ю., група КП-61

Додаток 2

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ТА ВИКОРИСТАННЯ ПЕРСОНАЛЬНИХ ДОКУМЕНТІВ

Виконав: студент групи КП-61 Згуровський Ярослав Юрійович

Науковий керівник: ст. викл. Гадиняк Руслан Анатолійович

Київ – 2020

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розроблення програмного забезпечення для зберігання персональних документів та їх використання, шляхом автозаповнення форм даними.

Завдання:

1. Проаналізувати предметну область, існуючі аналоги та моделі зберігання даних.
2. Розробити програмне забезпечення для зберігання персональних документів та сервіс автозаповнення.
3. Протестувати розроблені програмні засоби.

АКТУАЛЬНІСТЬ

- Зменшення витрат часу на пошук потрібних для заповнення форм документів
- Автоматизоване заповнення форм персональними даними
- Надійно захищене електронне сховище особистих документів

АНАЛІЗ МОДЕЛЕЙ ЗБЕРІГАННЯ ДАНИХ

1. Локальна
2. Хмарна
3. Solid
4. Гібридна ✓

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1. Dropbox
2. Sync.com
3. Tresorit

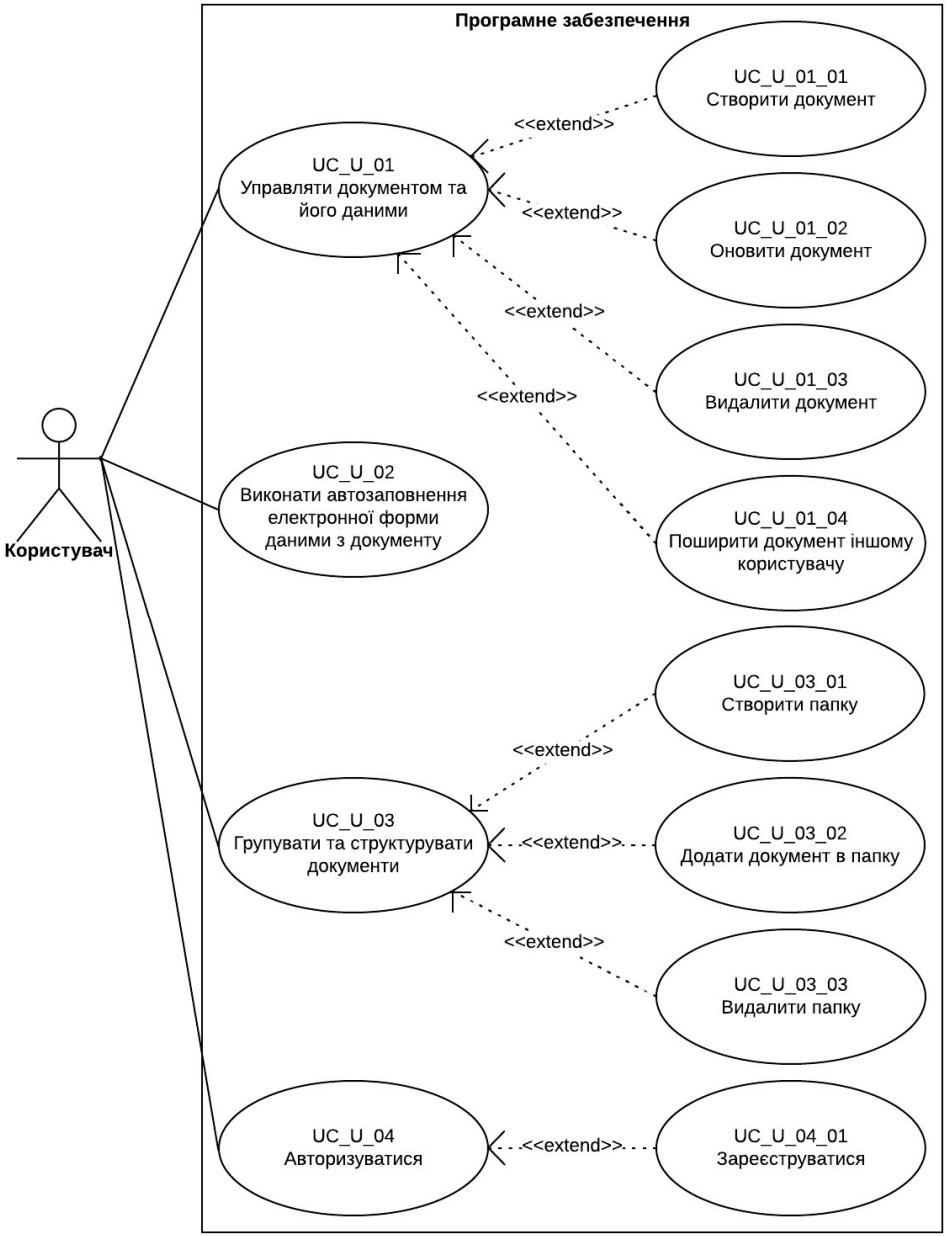
сервіси зберігання документів

4. LastPass менеджер паролей з можливістю автозаповнення

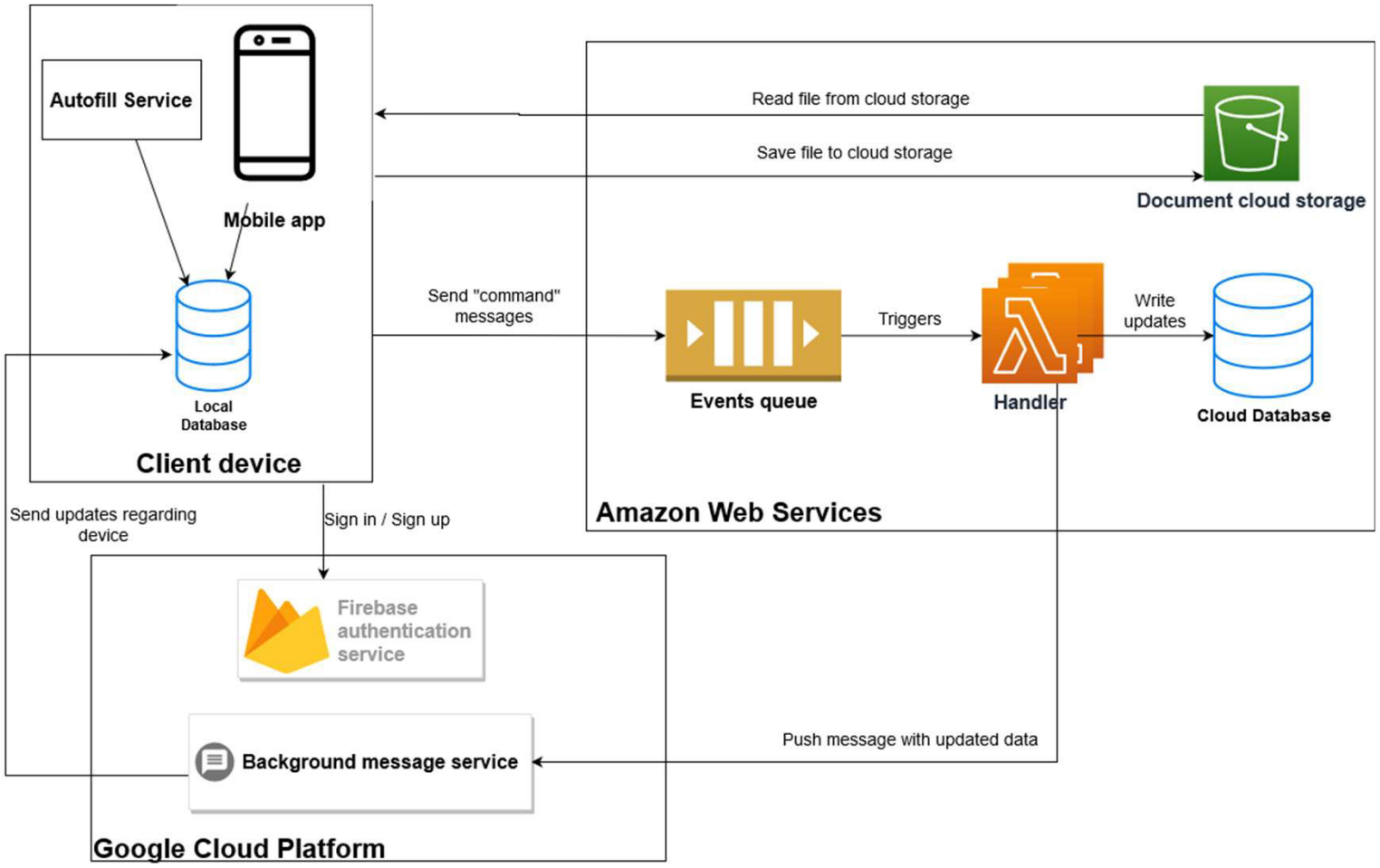
ВИМОГИ ДО ФУНКЦІОНАЛЬНОСТІ

- реєстрація нового користувача та автентифікація існуючого;
- завантаження в хмарне сховище персональних документів в електронному вигляді;
- скачування з хмарного сховища персональних документів в електронному вигляді;
- збереження та редагування персональних даних у сховищі;
- автоматизоване заповнення форм персональними даними користувача;
- обмін персональними даними з іншими користувачами системи.

ДІАГРАМА ПРЕЦЕДЕНТІВ



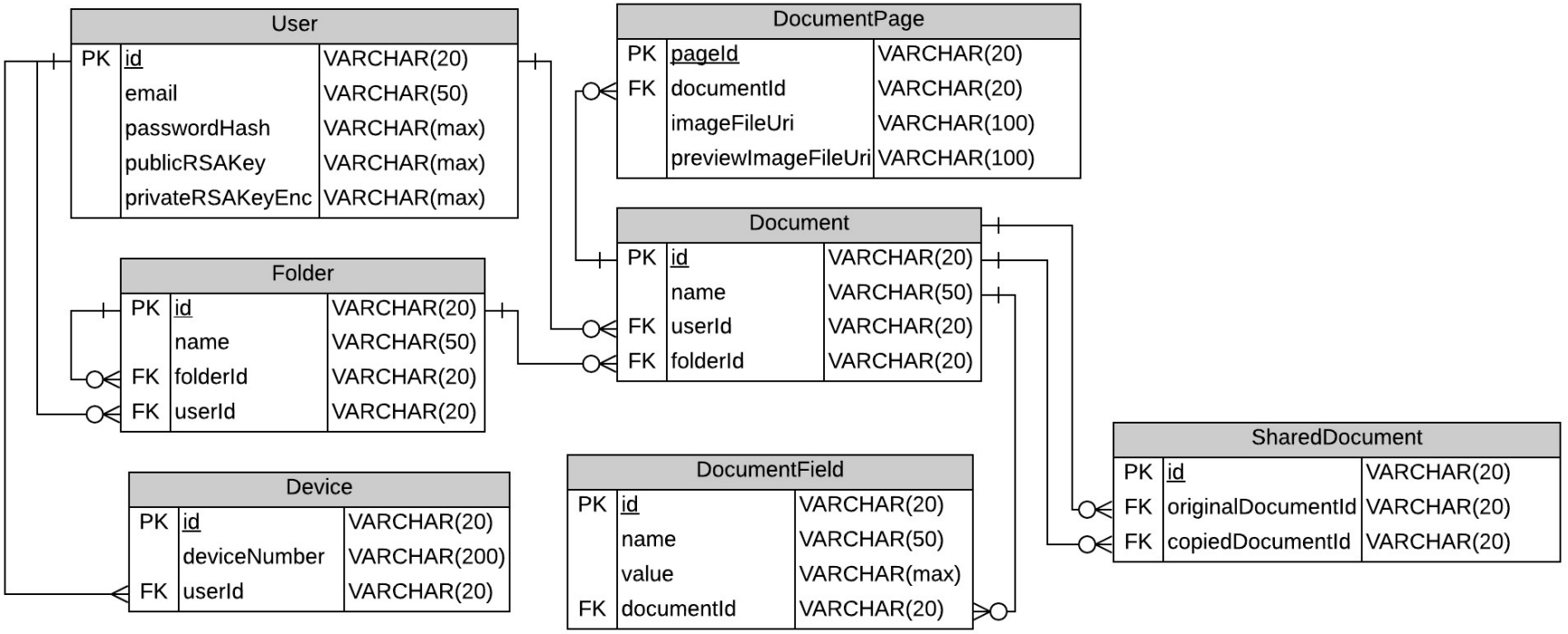
АРХІТЕКТУРА СИСТЕМИ



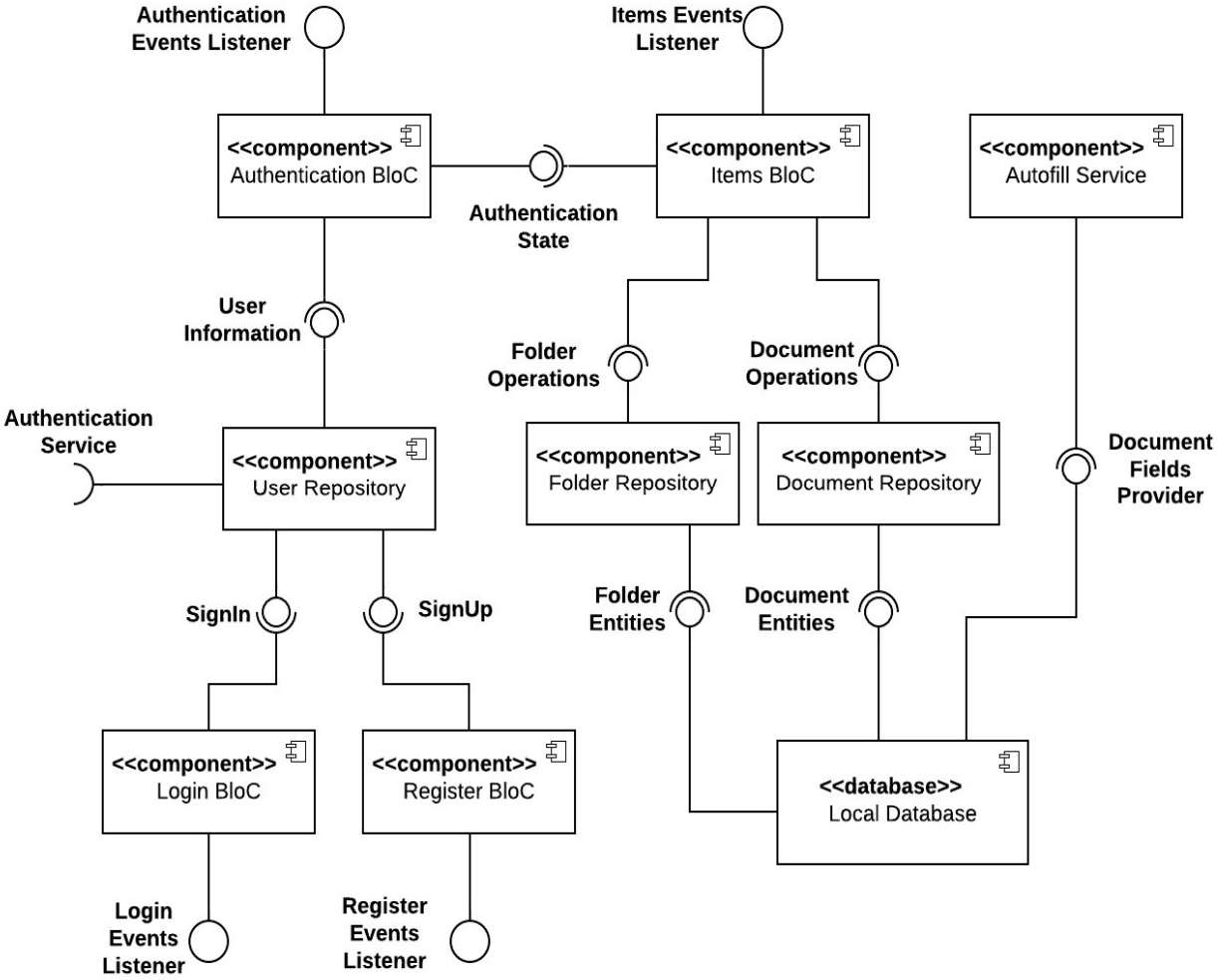
ЗАСОБИ ТА ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ

- Клієнтська частина:
  - Фреймворк для розробки мобільних застосунків – **Flutter**.
  - Мова програмування мобільного застосунку – **Dart**.
  - Локальна база даних – **SQLite**.
  - Сервіс автентифікації – **Firebase Authentication**.
- Серверна частина:
  - Серверний обробник – **AWS Lambda**.
  - Мова програмування серверного обробника – **C#**.
  - Хмарна база даних – **PostgreSQL**.
  - Хмарне сховище зображень документів – **Amazon S3**.
- Сервіси комунікації:
  - Сервіс повідомлень для мобільних пристроїв – **Firebase Cloud Messaging**.
  - Черга повідомлень для серверної частини – **AWS SQS**.

СТРУКТУРА БАЗИ ДАНИХ



ДІАГРАМА КОМПОНЕНТІВ КЛІЄНТСЬКОГО ЗАСТОСУНКУ



АЛГОРИТМ ОБРОБКИ ПОВІДОМЛЕНЬ ІЗ ЧЕРГИ

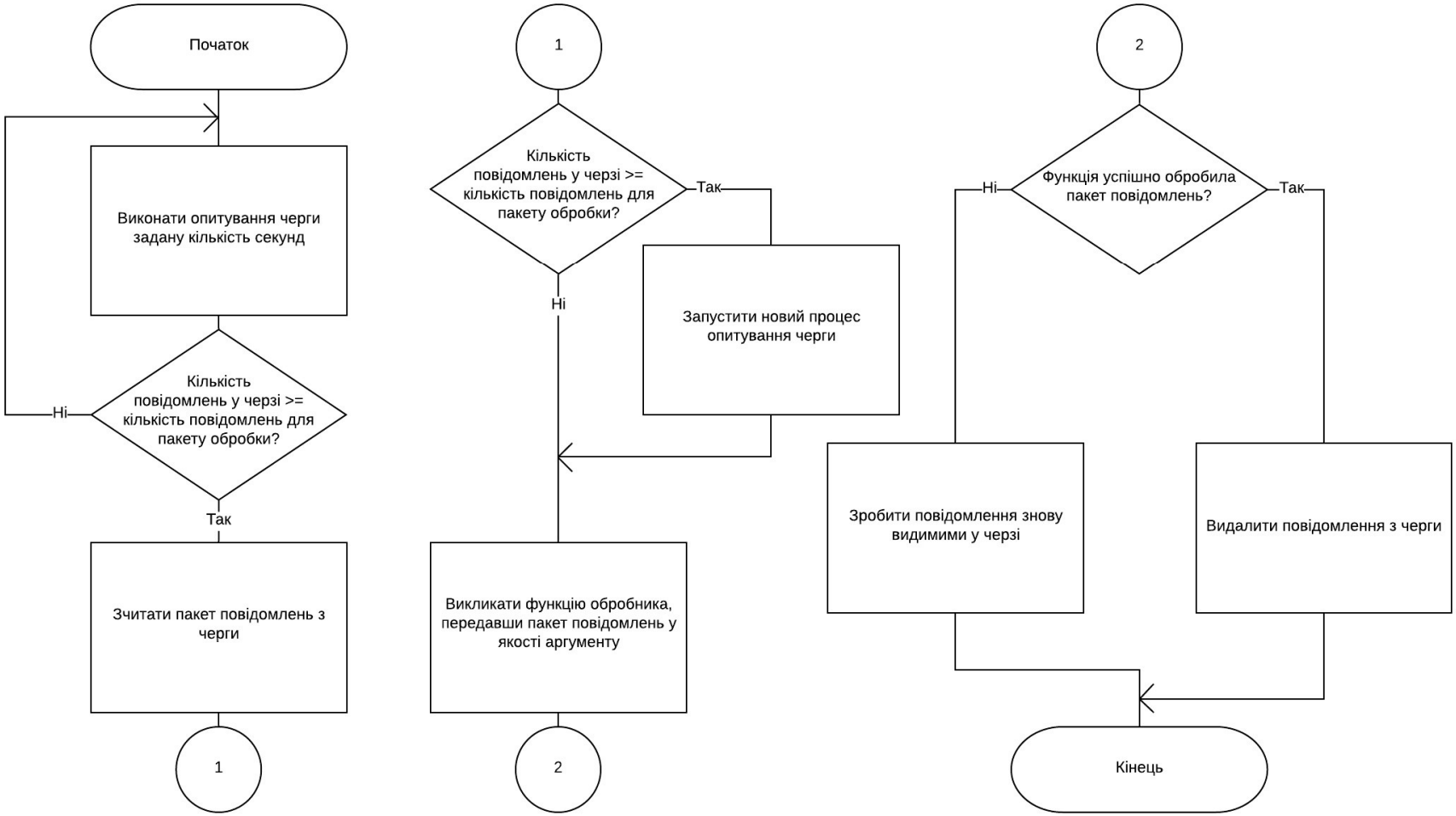
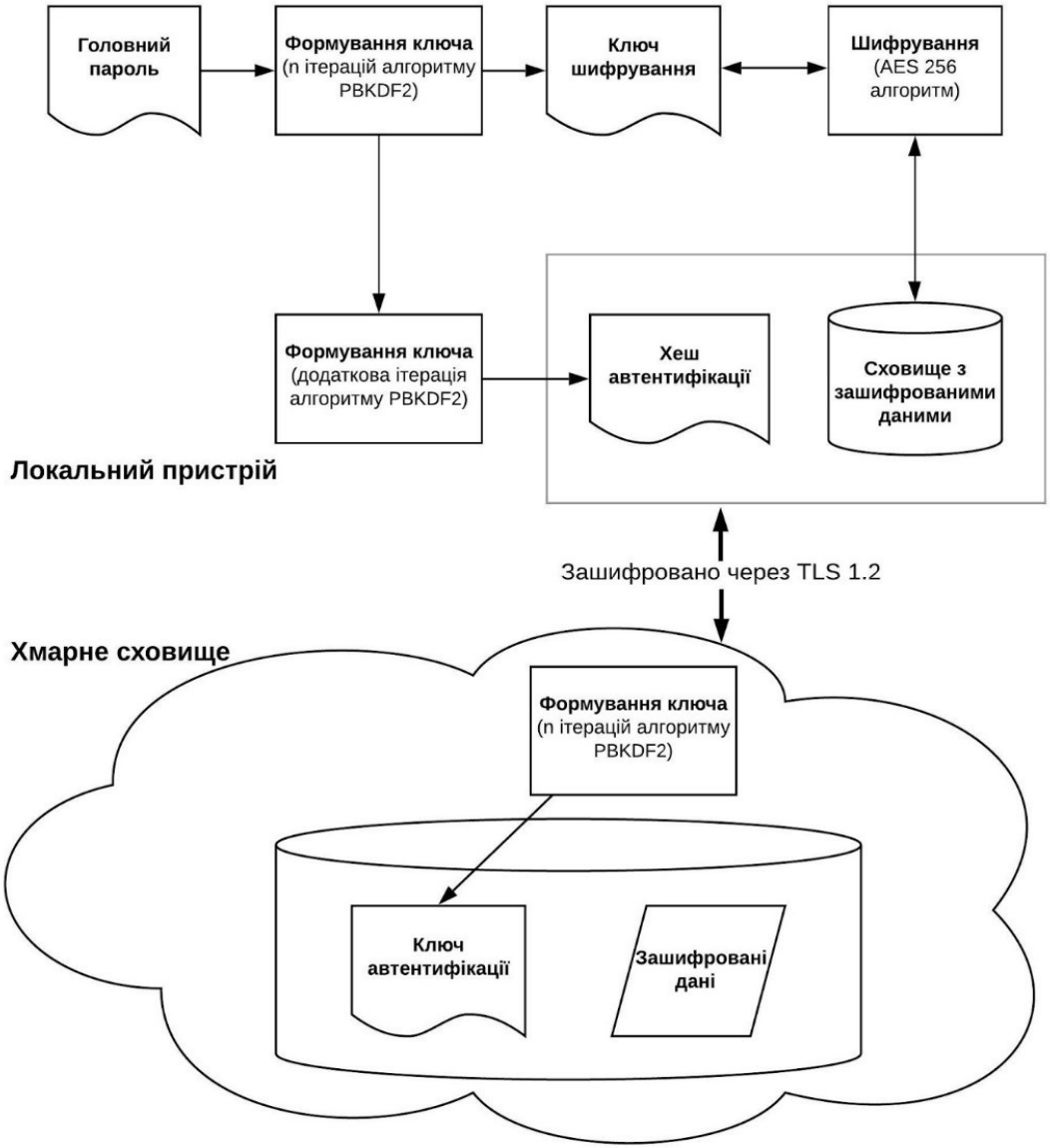
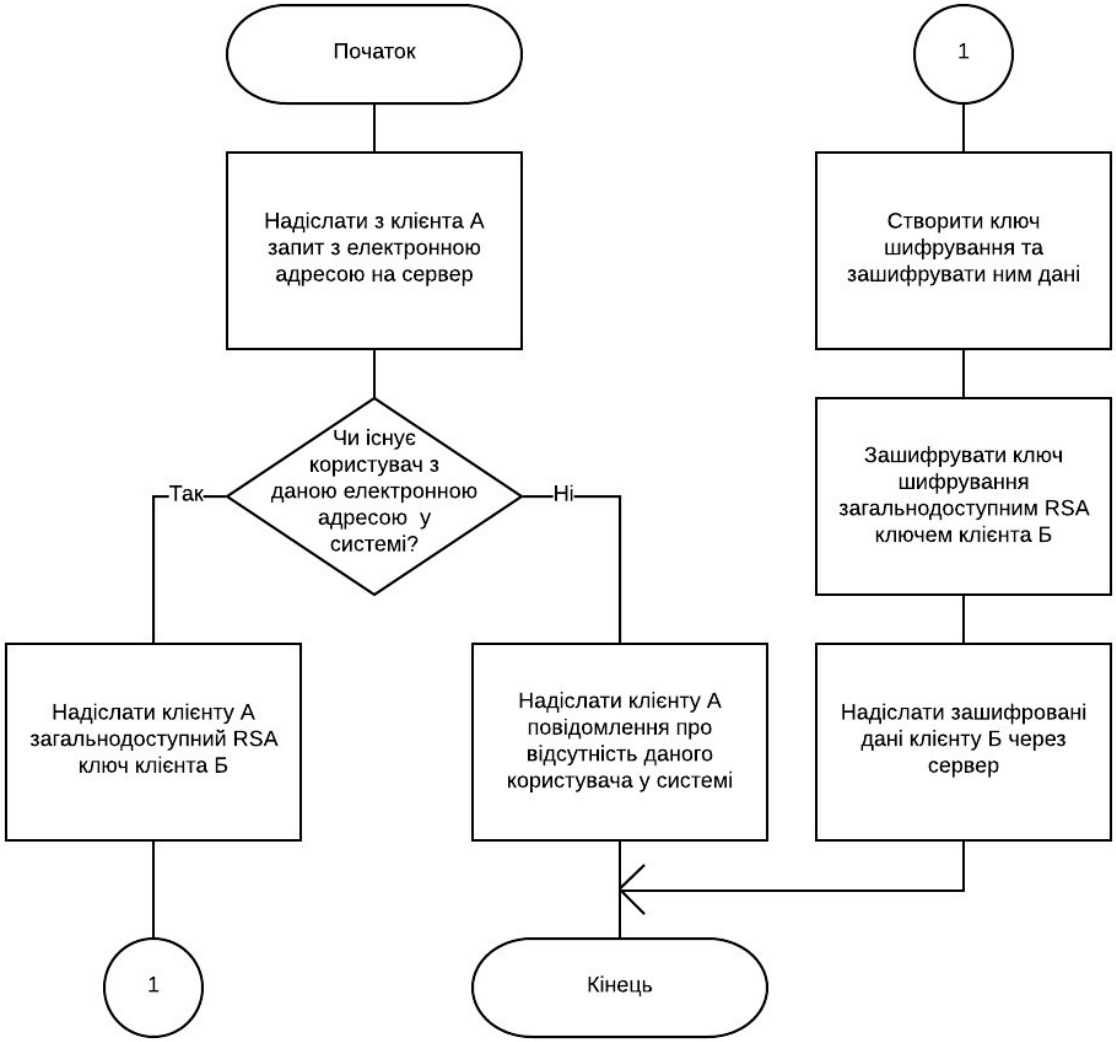


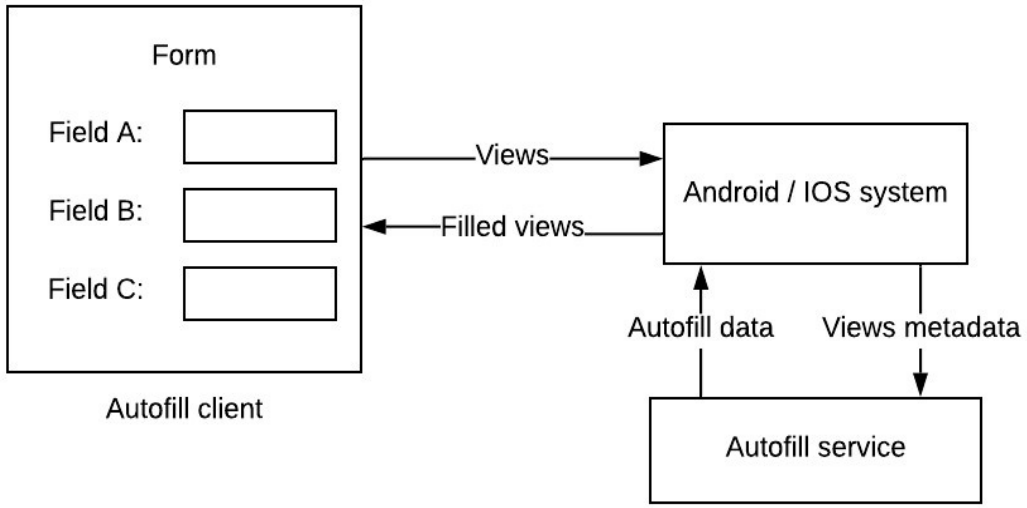
СХЕМА ПРОЦЕСУ ШИФРУВАННЯ ДАНИХ

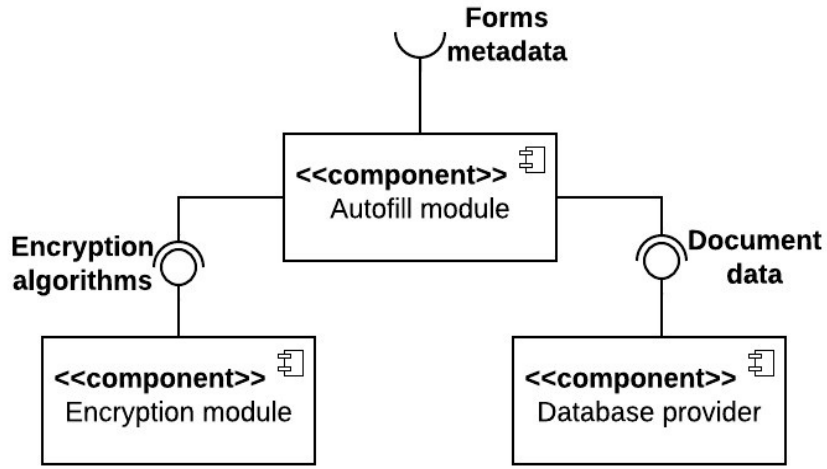


АЛГОРИТМ ШИФРУВАННЯ ДАНИХ ДЛЯ ПОШИРЕННЯ

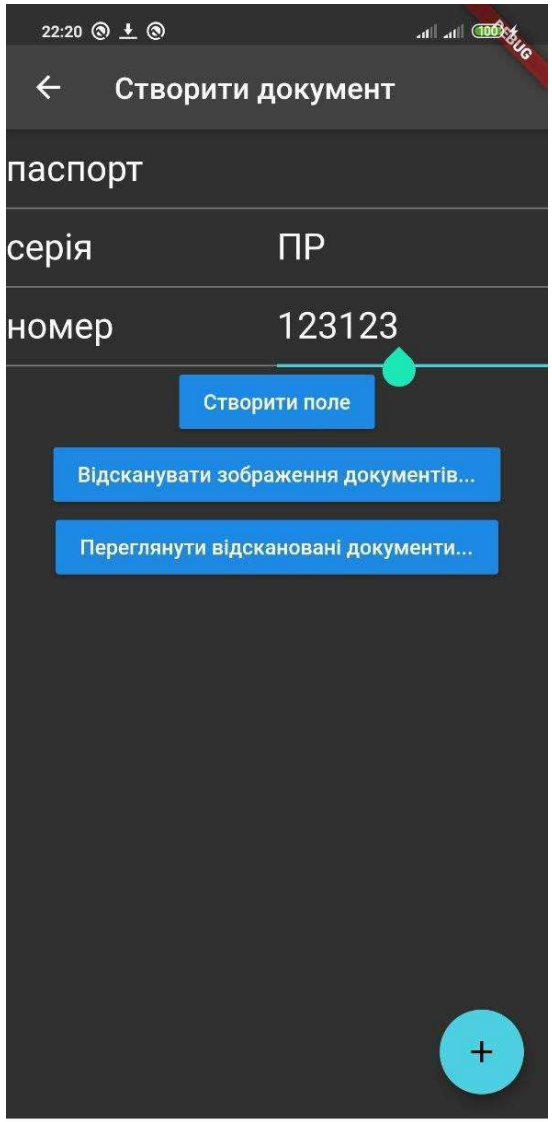


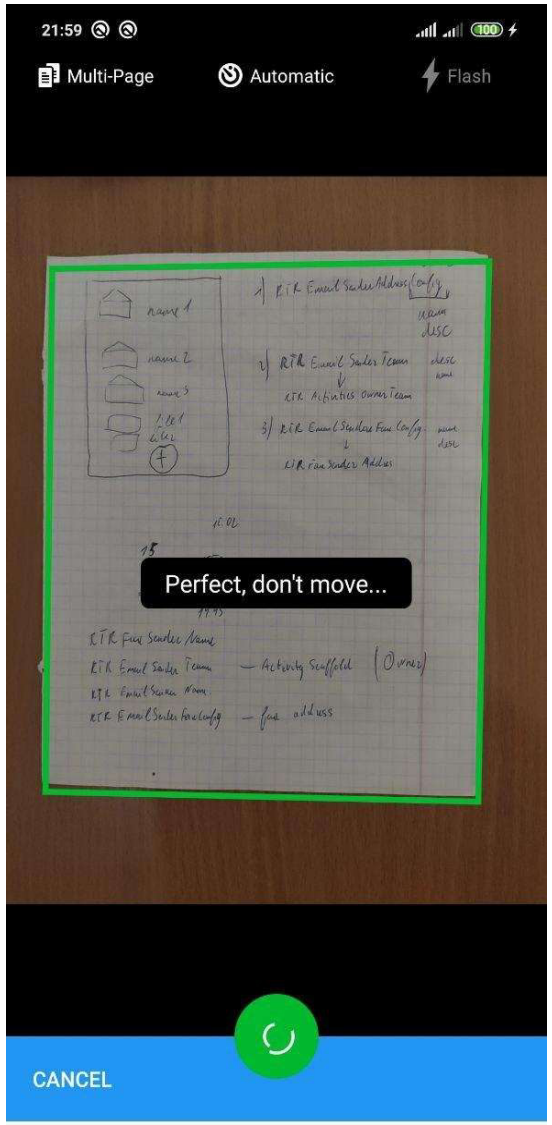
СЕРВІС АВТОЗАПОВНЕННЯ

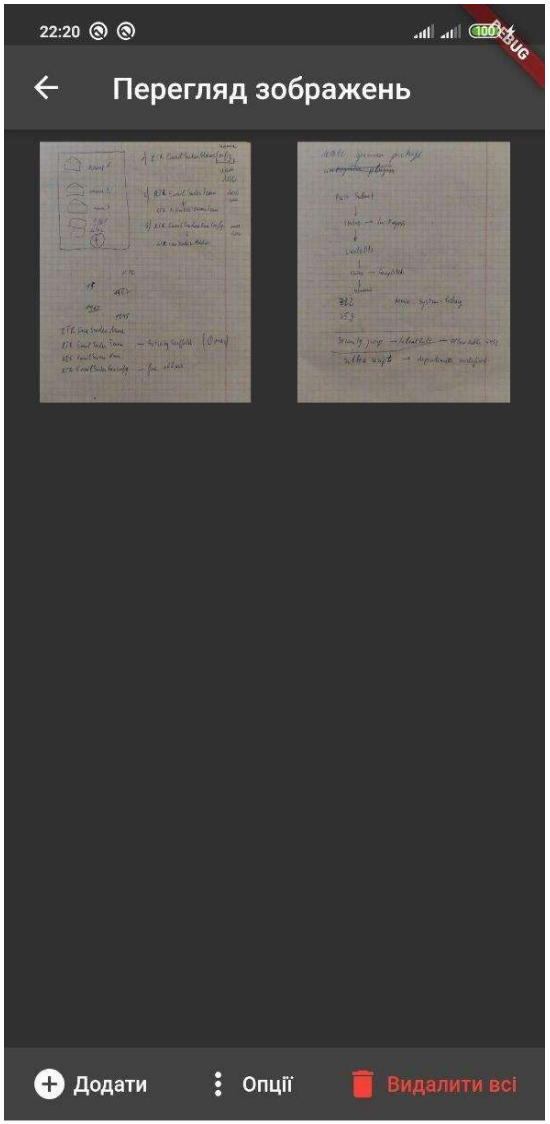




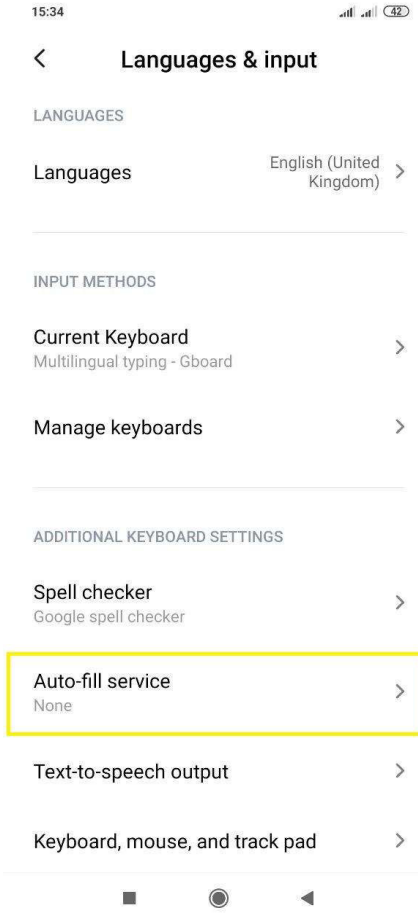
ІНТЕРФЕЙС ЗАСТОСУНКУ. СТВОРЕННЯ ДОКУМЕНТУ

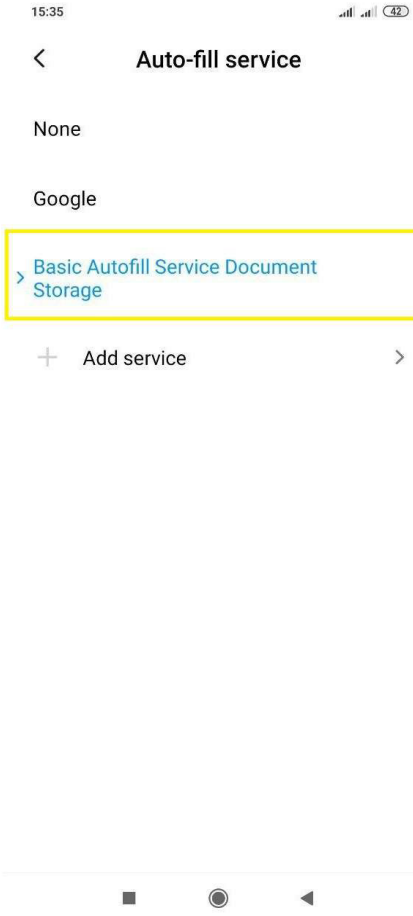


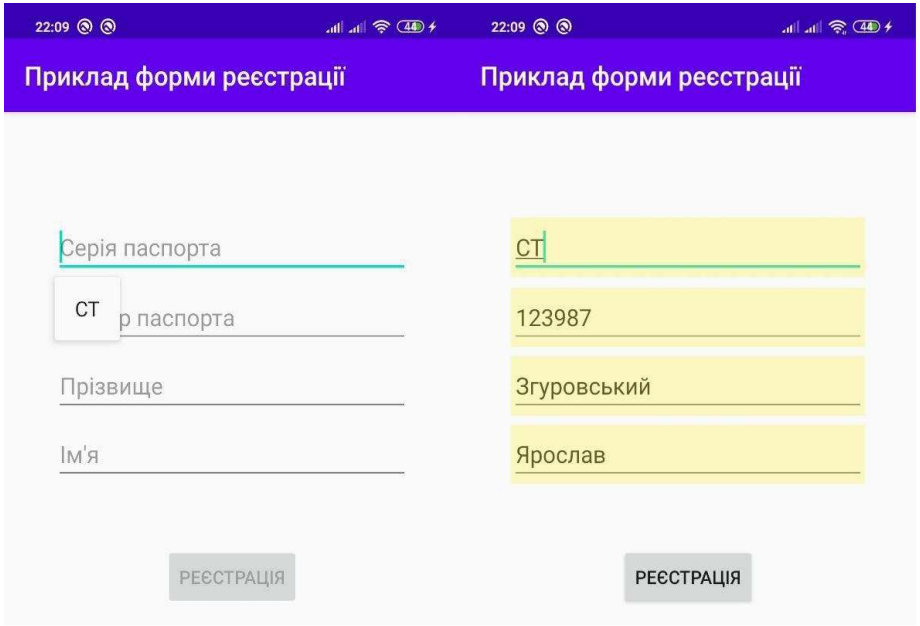




НАЛАШТУВАННЯ АВТОЗАПОВНЕННЯ ТА ПРИКЛАД РОБОТИ







ТЕСТУВАННЯ

1. Тестування виконується відповідно до техніки Gray Box Testing.
2. Методи тестування:
   - Тестування відповідності;
   - Функціональне тестування:
     - integration testing;
     - interface testing.
   - Нефункціональне тестування:
     - reliability testing;
     - efficiency testing.

ПОРІВНЯННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ З АНАЛОГАМИ

Критерій\Назва	DropBox	Sync.com	Tresorit	LastPass	Розроблений застосунок
Автозаповнення	−	−	−	+	+
Зберігання та управління документами	+	+	+	−	+
Шифрування на стороні клієнта	−	+	+	+	+

ШЛЯХИ ПОКРАЩЕННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

- Використання часткового шифрування бінарних файлів скан-копій замість повного
- Застосування технології оптичного розпізнавання символів для заповнення даних із документів.

ПЕРЕВІРКА НА УНІКАЛЬНІСТЬ

File name: **Згуровський_основний_ПЗ**

File ID: **1004047603** Page count: **48** Word count: **9006** Character count: **70306** File size: **409.95 KB**

3.06% Matches

Highest match: **0.71%** with library source. File ID: **1000083040**

0.46% Internet Matches	5	Page 50
2.95% Library matches	81	Page 50

0.52% Quotes

Quotes	2	Page 51

Exclude references is off

0% Exclusions

No exclusions found

Replacement

No replaced characters found

ВИСНОВКИ

1. Проведений аналіз наявних рішень виявив потребу створення даного програмного продукту.
2. Розроблене програмне забезпечення:
   - надає можливість здійснювати основні операції над документами, такі як створення, оновлення, видалення;
   - дозволяє ділитися доступом до власних документів іншим користувачам системи;
   - забезпечує надійний захист та конфіденційність персональних даних користувачів;
   - надає можливість автозаповнення форм персональними даними.
3. Розробку виконано в повному обсязі у відповідності до сформованих вимог.
4. Тестування продукту виконано відповідно до затвердженої програми та методики тестування.

Дякую за увагу!

Додаток 3

Лістинг програми

```
import 'dart:async';
import 'package:bloc/bloc.dart';
import 'package:equatable/equatable.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'package:flutter_todos/repositories/user_repository.dart';
import 'package:meta/meta.dart';

part 'auth_event.dart';
part 'auth_state.dart';

class AuthenticationBloc
    extends Bloc<AuthenticationEvent, AuthenticationState> {
  final UserRepository _userRepository;
  final storage = new FlutterSecureStorage();

  AuthenticationBloc({@required UserRepository userRepository})
      : assert(userRepository != null),
        _userRepository = userRepository;

  @override
  AuthenticationState get initialState => Uninitialized();

  @override
  Stream<AuthenticationState> mapEventToState(
    AuthenticationEvent event,
  ) async* {
    if (event is AppStarted) {
      yield* _mapAppStartedToState();
    } else if (event is LoggedIn) {
      yield* _mapLoggedInToState();
    } else if (event is LoggedOut) {
      yield* _mapLoggedOutToState();
    }
  }

  Stream<AuthenticationState> _mapAppStartedToState() async* {
    final isSignedIn = await _userRepository.isSignedIn();
    if (isSignedIn) {
      final name = await _userRepository.getUser();
      yield Authenticated(name);
    } else {
      yield Unauthenticated();
    }
  }

  Stream<AuthenticationState> _mapLoggedInToState() async* {
    yield Authenticated(await _userRepository.getUser());
  }

  Stream<AuthenticationState> _mapLoggedOutToState() async* {
    yield Unauthenticated();
    _userRepository.signOut();
    storage.delete(key: 'userId');
    storage.delete(key: 'userKey');
  }
}

import 'dart:async';
import 'package:bloc/bloc.dart';

import 'document_fields.dart';

class DocumentFieldsBloc extends Bloc<DocumentFieldsEvent,
DocumentFieldsState> {
```

```
  @override
  DocumentFieldsState get initialState => DocumentFieldsLoad({});

  @override
  Stream<DocumentFieldsState> mapEventToState(DocumentFieldsEvent event)
async* {
    yield* _mapEmptyFieldAddedToState();
  }

  Stream<DocumentFieldsState> _mapEmptyFieldAddedToState() async* {
    Map<String, String> fields = (state as DocumentFieldsLoad).fields;

    fields.putIfAbsent("", () => "");

    yield DocumentFieldsLoad(fields);
  }
}
```

```
import 'items_event.dart';
import 'items_state.dart';

class ItemsBloc extends Bloc<ItemsEvent, ItemsState> {
  final AuthenticationBloc authBloc;
  StreamSubscription authSubscription;
  final DocumentsRepository documentsRepository;
  final FoldersRepository foldersRepository;

  ItemsBloc({@required this.documentsRepository, @required
this.foldersRepository, @required this.authBloc}){
    authSubscription = authBloc.listen((state) {
      if (state is Authenticated) {
        add(ItemsLoaded(null, true));
      }
    });
  }

  @override
  ItemsState get initialState => ItemsLoadInProgress();

  @override
  Stream<ItemsState> mapEventToState(ItemsEvent event) async* {
    if (event is ItemsLoaded) {
      yield* _mapItemsLoadedToState(event);
    } else if (event is DocumentAdded) {
      yield* _mapDocumentAddedToState(event);
    } else if (event is DocumentUpdated) {
      yield* _mapDocumentUpdatedToState(event);
    } else if (event is DocumentDeleted) {
      yield* _mapDocumentDeletedToState(event);
    }  else if (event is FolderAdded) {
      yield* _mapFolderAddedToState(event);
    } else if (event is FolderUpdated) {
      yield* _mapFolderUpdatedToState(event);
    } else if (event is FolderDeleted) {
      yield* _mapFolderDeletedToState(event);
    } else if(event is DocumentLoaded) {
      yield* _mapDocumentLoadedToState(event);
    }
  }

  Stream<ItemsState> _mapItemsLoadedToState(ItemsLoaded event) async* {
    try {
      final folder = event.folder;
```

```
      final docs = await
this.documentsRepository.loadChildDocuments(folder?.id);
      final folders = await
this.foldersRepository.loadChildFolders(folder?.id);

      yield ItemsLoadSuccess(
        folder,
        docs.where((i) => i.isOwn ==
event.isOwn).map(Document.fromEntity).toList(),
        folders.where((f) => f.isOwn ==
event.isOwn).map(Folder.fromEntity).toList(),
      );
    } catch (_) {
      yield ItemsLoadFailure();
    }
  }

  Stream<ItemsState> _mapDocumentAddedToState(DocumentAdded event) async*
{
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final newDocument = event.document;
      final List<Document> updatedDocuments =
List.from(currentState.documents)
        ..add(newDocument);
      yield ItemsLoadSuccess(currentState.currentFolder, updatedDocuments,
currentState.folders);
      List<DocumentFieldEntity> fields = event.fields.entries.map((f) =>
DocumentField(newDocument.id, f.key, f.value).toEntity()).toList();
      documentsRepository.createDocument(event.document.toEntity(),
fields);
    }
  }

  Stream<ItemsState> _mapDocumentUpdatedToState(DocumentUpdated event)
async* {
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final List<Document> updatedDocuments =
currentState.documents.map((doc) {
        return doc.id == event.document.id ? event.document : doc;
      }).toList();
      yield ItemsLoadSuccess(currentState.folders.firstWhere((f) => f.id
== event.document.parentFolderId), updatedDocuments,
currentState.folders);
      _saveDocuments(updatedDocuments);
    }
  }

  Stream<ItemsState> _mapDocumentDeletedToState(DocumentDeleted event)
async* {
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final updatedDocuments = currentState
          .documents
          .where((doc) => doc.id != event.document.id)
          .toList();
      yield ItemsLoadSuccess(currentState.folders.firstWhere((f) => f.id
== event.document.parentFolderId), updatedDocuments,
currentState.folders);
      _saveDocuments(updatedDocuments);
    }
  }
```

```
  Stream<ItemsState> _mapFolderAddedToState(FolderAdded event) async* {
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final List<Folder> updatedFolders = List.from(currentState.folders)
        ..add(event.folder);

      yield ItemsLoadSuccess(currentState.currentFolder,
currentState.documents, updatedFolders);
      foldersRepository.createFolder(event.folder.toEntity());
    }
  }

  Stream<ItemsState> _mapFolderUpdatedToState(FolderUpdated event) async*
{
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final List<Folder> updatedFolders = currentState.folders.map((f) {
        return f.id == event.folder.id ? event.folder : f;
      }).toList();
      yield ItemsLoadSuccess(updatedFolders?.firstWhere((f) => f.id ==
event.folder.parentFolderId),currentState.documents, updatedFolders);
      _saveFolders(updatedFolders);
    }
  }

  Stream<ItemsState> _mapFolderDeletedToState(FolderDeleted event) async*
{
    if (state is ItemsLoadSuccess) {
      var currentState = state as ItemsLoadSuccess;
      final List<Folder> updatedFolders = currentState
          .folders
          .where((f) => f.id != event.folder.id)
          .toList();
      yield ItemsLoadSuccess(updatedFolders?.firstWhere((f) => f.id ==
event.folder.parentFolderId),currentState.documents, updatedFolders);
      _saveFolders(updatedFolders);
    }
  }

  Stream<ItemsState> _mapDocumentLoadedToState(DocumentLoaded event)
async* {
    var document = await
documentsRepository.getDocument(event.documentId);
    yield DocumentLoadSuccess(Document.fromEntity(document));
  }

  Future _saveDocuments(List<Document> docs) {
    return documentsRepository.saveDocuments(
      docs.map((doc) => doc.toEntity()).toList(),
    );
  }

  Future _saveFolders(List<Folder> folders) {
    return foldersRepository.saveFolders(
      folders.map((f) => f.toEntity()).toList(),
    );
  }
}

import 'login_event.dart';
import 'login_state.dart';

class LoginBloc extends Bloc<LoginEvent, LoginState> {
  UserRepository _userRepository;
```

```
  final storage = new FlutterSecureStorage();

  LoginBloc({
    @required UserRepository userRepository,
  })  : assert(userRepository != null),
        _userRepository = userRepository;

  @override
  LoginState get initialState => LoginState.empty();

  @override
  Stream<LoginState> transformEvents(
    Stream<LoginEvent> events,
    Stream<LoginState> Function(LoginEvent) next,
  ) {
    final nonDebounceStream = events.where((event) {
      return (event is! EmailChanged && event is! PasswordChanged);
    });
    final debounceStream = events.where((event) {
      return (event is EmailChanged || event is PasswordChanged);
    }).debounceTime(Duration(milliseconds: 300));
    return super.transformEvents(
      nonDebounceStream.mergeWith([debounceStream]),
      next,
    );
  }

  @override
  Stream<LoginState> mapEventToState(LoginEvent event) async* {
    if (event is EmailChanged) {
      yield* _mapEmailChangedToState(event.email);
    } else if (event is PasswordChanged) {
      yield* _mapPasswordChangedToState(event.password);
    } else if (event is LoginWithCredentialsPressed) {
      yield* _mapLoginWithCredentialsPressedToState(
        email: event.email,
        password: event.password,
      );
    }
  }

  Stream<LoginState> _mapEmailChangedToState(String email) async* {
    yield state.update(
      isEmailValid: Validators.isValidEmail(email),
    );
  }

  Stream<LoginState> _mapPasswordChangedToState(String password) async* {
    yield state.update(
      isPasswordValid: Validators.isValidPassword(password),
    );
  }

  Stream<LoginState> _mapLoginWithCredentialsPressedToState({
    String email,
    String password,
  }) async* {
    yield LoginState.loading();
    try {
      var authResult = await _userRepository.signInWithCredentials(email,
password);
      await storage.write(key: 'userId', value: authResult.user.uid);
      final hash = Password.hash(password, PBKDF2(desiredKeyLength: 256));
      await storage.write(key: 'userKey', value: hash);
```

```
      yield LoginState.success();
    } catch (_) {
      yield LoginState.failure();
    }
  }
}

class RegisterBloc extends Bloc<RegisterEvent, RegisterState> {
  final UserRepository _userRepository;
  final SqsService _sqsService;
  final storage = new FlutterSecureStorage();

  RegisterBloc(
      {@required UserRepository userRepository,
      @required SqsService sqsService})
      : assert(userRepository != null),
        assert(sqsService != null),
        _userRepository = userRepository,
        _sqsService = sqsService;

  @override
  RegisterState get initialState => RegisterState.empty();

  @override
  Stream<RegisterState> transformEvents(
    Stream<RegisterEvent> events,
    Stream<RegisterState> Function(RegisterEvent) next,
  ) {
    final nonDebounceStream = events.where((event) {
      return (event is! EmailChanged && event is! PasswordChanged);
    });
    final debounceStream = events.where((event) {
      return (event is EmailChanged || event is PasswordChanged);
    }).debounceTime(Duration(milliseconds: 300));
    return super.transformEvents(
      nonDebounceStream.mergeWith([debounceStream]),
      next,
    );
  }

  @override
  Stream<RegisterState> mapEventToState(
    RegisterEvent event,
  ) async* {
    if (event is EmailChanged) {
      yield* _mapEmailChangedToState(event.email);
    } else if (event is PasswordChanged) {
      yield* _mapPasswordChangedToState(event.password);
    } else if (event is Submitted) {
      yield* _mapFormSubmittedToState(event.email, event.password);
    }
  }

  Stream<RegisterState> _mapEmailChangedToState(String email) async* {
    yield state.update(
      isEmailValid: Validators.isValidEmail(email),
    );
  }

  Stream<RegisterState> _mapPasswordChangedToState(String password) async*
{
    yield state.update(
      isPasswordValid: Validators.isValidPassword(password),
    );
```

```
  }

  Stream<RegisterState> _mapFormSubmittedToState(
    String email,
    String password,
  ) async* {
    yield RegisterState.loading();
    try {
      var authResult = await _userRepository.signUp(
        email: email,
        password: password,
      );
      await storage.write(key: 'userId', value: authResult.user.uid);

      final hash = Password.hash(password, PBKDF2(desiredKeyLength: 256));
      await storage.write(key: 'userKey', value: hash);

      yield RegisterState.success();

      var sendMessageResult = await _sqsService.sendMessage(
          MessageType.USER_CREATED,
          {"user_id": authResult.user.uid, "pass_hash": hash}.toString());
      print(sendMessageResult.toString());
    } catch (_) {
      yield RegisterState.failure();
    }
  }
}

class TabBloc extends Bloc<TabEvent, AppTab> {
  @override
  AppTab get initialState => AppTab.own;

  @override
  Stream<AppTab> mapEventToState(TabEvent event) async* {
    if (event is TabUpdated) {
      yield event.tab;
    }
  }
}

typedef OnSaveCallback = Function(String name);

class AddEditFolderScreen extends StatefulWidget {
  final bool isEditing;
  final Folder folder;

  AddEditFolderScreen({
    Key key,
    @required this.isEditing,
    this.folder,
  }) : super(key: key ?? Keys.addFolderScreen);

  @override
  _AddEditFolderScreenState createState() => _AddEditFolderScreenState();
}

class _AddEditFolderScreenState extends State<AddEditFolderScreen> {
  static final GlobalKey<FormState> _formKey = GlobalKey<FormState>();

  String _name;

  bool get isEditing => widget.isEditing;
  Folder get folder => widget.folder;
```

```
  @override
  Widget build(BuildContext context) {
    final localizations = SampleLocalizations.of(context);
    final textTheme = Theme.of(context).textTheme;

    return BlocBuilder<ItemsBloc, ItemsState>(builder: (context,
activeTab) {
      return Scaffold(
        appBar: AppBar(
          title: Text(
            isEditing
                ? "Редагувати папку"
                : "Створити папку", //localizations.editFolder :
localizations.addFolder,
          ),
        ),
        body: Padding(
          padding: EdgeInsets.all(16.0),
          child: Form(
            key: _formKey,
            child: ListView(
              children: [
                TextFormField(
                  initialValue: isEditing ? widget.folder.name : '',
                  key: Keys.nameFolderField,
                  autofocus: !isEditing,
                  style: textTheme.headline,
                  decoration: InputDecoration(
                    hintText: "Назва папки"//localizations.newTodoHint,
                  ),
                  validator: (val) {
                    return val.trim().isEmpty
                        ? localizations.emptyTodoError
                        : null;
                  },
                  onSaved: (value) => _name = value,
                ),
              ],
            ),
          ),
        ),
        floatingActionButton: FloatingActionButton(
          key: isEditing ? Keys.saveFolderFab : Keys.saveNewFolder,
          tooltip:
              isEditing ? localizations.saveChanges :
localizations.addFolder,
          child: Icon(isEditing ? Icons.check : Icons.add),
          onPressed: () {
            print("PRESSED");
            if (_formKey.currentState.validate()) {
              _formKey.currentState.save();

              if (isEditing) {
                BlocProvider.of<ItemsBloc>(context).add(
                  FolderUpdated(
                    folder.copyWith(name: _name),
                  ),
                );
              } else {
                var authState =
BlocProvider.of<AuthenticationBloc>(context).state;
```

```
                var currentState =
BlocProvider.of<ItemsBloc>(context).state;
                if (currentState is ItemsLoadSuccess) {

BlocProvider.of<ItemsBloc>(context).add(FolderAdded(Folder(
                      _name, true, (authState as Authenticated).userId,
                      parentFolderId: currentState.currentFolder?.id)));
                }
              }

              Navigator.pop(context);
              Navigator.pop(context);
            }
          },
        ),
      );
    });
  }
}


class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final state = BlocProvider.of<ItemsBloc>(context).state;
    Folder folder;
    if (state is ItemsLoadSuccess) {
      folder = state.currentFolder;
    }
    return BlocBuilder<TabBloc, AppTab>(
      builder: (context, activeTab) {
        return Scaffold(
          appBar: AppBar(
            title: AppBarText(),
            actions: <Widget>[
              IconButton(
                icon: Icon(Icons.exit_to_app),
                onPressed: () {
                  BlocProvider.of<AuthenticationBloc>(context).add(
                    LoggedOut(),
                  );
                },
              )
            ],
            leading: AppBarLeading(),
          ),
          body: BlocProvider.of<TabBloc>(context).state == AppTab.own
              ? ItemsList(folder?.id, true)
              : ItemsList(folder?.id, false),
          floatingActionButton: Visibility(
            visible: BlocProvider.of<TabBloc>(context).state ==
AppTab.own,
            child: FloatingActionButton(
              key: Keys.addItemFab,
              onPressed: () {
                _settingModalBottomSheet(context);
              },
              child: Icon(Icons.add),
              tooltip: SampleLocalizations.of(context).addFolder,
            ),
          ),
          bottomNavigationBar: TabSelector(
            activeTab: activeTab,
            onTabSelected: (tab) =>
```

```
                BlocProvider.of<TabBloc>(context).add(TabUpdated(tab)),
          ),
        );
      },
    );
  }

  void _settingModalBottomSheet(context) {
    showModalBottomSheet(
        context: context,
        builder: (BuildContext bc) {
          return Container(
            child: new Wrap(
              children: <Widget>[
                new Container(
                  child: Center(
                    child: Text(
                      "Створити",
                      style: TextStyle(
                        fontSize: 20,
                      ),
                    ),
                  ),
                  margin: new EdgeInsets.all(5),
                ),
                new ListTile(
                    leading: new Icon(Icons.folder),
                    title: new Text('Папку'),
                    onTap: () =>
                        {Navigator.pushNamed(context, Routes.addFolder)}),
                new ListTile(
                  leading: new Icon(Icons.insert_drive_file),
                  title: new Text('Документ'),
                  onTap: () =>
                      {Navigator.pushNamed(context, Routes.addDocument)},
                ),
              ],
            ),
          );
        });
  }
}

class LoginScreen extends StatelessWidget {
  final UserRepository _userRepository;
  final SqsService _sqsService;

  LoginScreen({Key key, @required UserRepository userRepository, @required
SqsService sqsService})
      : assert(userRepository != null), assert(sqsService != null),
        _userRepository = userRepository,
        _sqsService = sqsService,
        super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Вхід')),
      body: BlocProvider<LoginBloc>(
        create: (context) => LoginBloc(userRepository: _userRepository),
        child: LoginForm(userRepository: _userRepository, sqsService:
_sqsService),
      ),
    );
```

```
  }
}


class RegisterScreen extends StatelessWidget {
  final UserRepository _userRepository;
  final SqsService _sqsService;

  RegisterScreen({Key key, @required UserRepository userRepository,
@required SqsService sqsService})
      : assert(userRepository != null), assert(sqsService != null),
        _userRepository = userRepository,
        _sqsService = sqsService,
        super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Реєстрація')),
      body: Center(
        child: BlocProvider<RegisterBloc>(
          create: (context) => RegisterBloc(userRepository:
_userRepository, sqsService: _sqsService),
          child: RegisterForm(),
        ),
      ),
    );
  }
}
```

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2019 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ТА ВИКОРИСТАННЯ ПЕРСОНАЛЬНИХ ДОКУМЕНТІВ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ярослав ЗГУРОВСЬКИЙ

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Програмне забезпечення для зберігання та використання персональних документів, що реалізоване у вигляді крос-платформного мобільного застосунку.

2. МЕТА ТЕСТУВАННЯ

Метою тестування є перевірка наступних елементів:

1. Доступ до хмарної бази даних через серверну частину.
2. Доступ до віддаленого файлового сховища.
3. Коректна взаємодія клієнтської частини із сервісом автентифікації.
4. Коректна взаємодія клієнтської частини з серверним обробником.
5. Відповідність інтерфейсу нефункціональним вимогам.
6. Забезпечення необхідного рівня безпеки даних.
7. Відповідність програмного забезпечення вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується відповідно до техніки Gray Box Testing, в ході якої на відповідність висунутим вимогам перевіряється як зовнішня поведінка системи, так і її внутрішня реалізація.

Використовуються наступні методи:

1. Тестування відповідності – тестування для перевірки відповідності функціональних можливостей додатку функціональним вимогам (acceptance testing).

2. Функціональне тестування, зокрема на рівні integration testing (тестування взаємодії різних модулів програми) та interface testing (тестування користувацької взаємодії).
3. Нефункціональне тестування, зокрема на рівні reliability testing (тестування надійності), usability testing (тестування зручності використання) та efficiency testing (тестування ефективності).

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність програмного забезпечення перевіряється шляхом:

1. Виконання юніт-тестів для серверної частини з використанням бібліотек NUnit та FluentAssertions.
2. Динамічного ручного тестування на відповідність функціональним вимогам.
3. Динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які доступні для використання у рамках даного програмного проєкту.
4. Статичного тестування коду.
5. Тестування мобільного застосунку на операційних системах Android та IOS.
6. Тестування зручності використання.
7. Тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБЕРІГАННЯ ТА ВИКОРИСТАННЯ ПЕРСОНАЛЬНИХ ДОКУМЕНТІВ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Руслан ГАДИНЯК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Ярослав ЗГУРОВСЬКИЙ

2020

ЗМІСТ

1. Опис структури інтерфейсу мобільного додатку

Клієнтська частина розробленого програмного забезпечення реалізована у вигляді крос-платформного мобільного застосунку і складається з кількох екранів, для роботи користувача з програмою.

Мобільний застосунок є одномовним, використовується українська мова. На екрані зйомки сторінок документу можуть з’являтись підказки англійською мовою, оскільки він використовує вбудовані віджети бібліотеки для сканування зображення документа.

Додаток включає наступні екрани:

1. Екран входу в застосунок.
2. Екран реєстрації нового користувача.
3. Екран-заставка.
4. Головний екран.
5. Екран відображення документа.
6. Екран для створення/редагування документа.
7. Екран для створення/редагування папки.
8. Екран для перегляду сторінок документу.
9. Екран для зйомки сторінок документу.

Кожен екран має верхню панель, що відображає на якому екрані зараз знаходиться користувач.

Якщо користувач авторизований в застосунку, то йому відкривається головний екран зі списком папок та файлів в кореневій папці.

Якщо користувач неавторизований в застосунку, то йому відкривається екран входу в застосунок. При цьому він має змогу перейти на екран реєстрації, якщо він не має існуючого профілю користувача.

2. Процедура автентифікації та реєстрації

На екрані входу відображено форму для автентифікації та кнопки, що дозволяють переходити на екран реєстрації, або головний екран застосунку. Доки користувач не заповнить поля відповідними валідними даними для автентифікації: електронну адресу та пароль користувача, кнопка входу в застосунок є неактивною. Поля форми валідуються за допомогою регулярних виразів. На рис. 2.1 та рис. 2.2 зображені приклади екрану та форми входу, що заповнена валідними та невалідними значеннями для полів вводу.

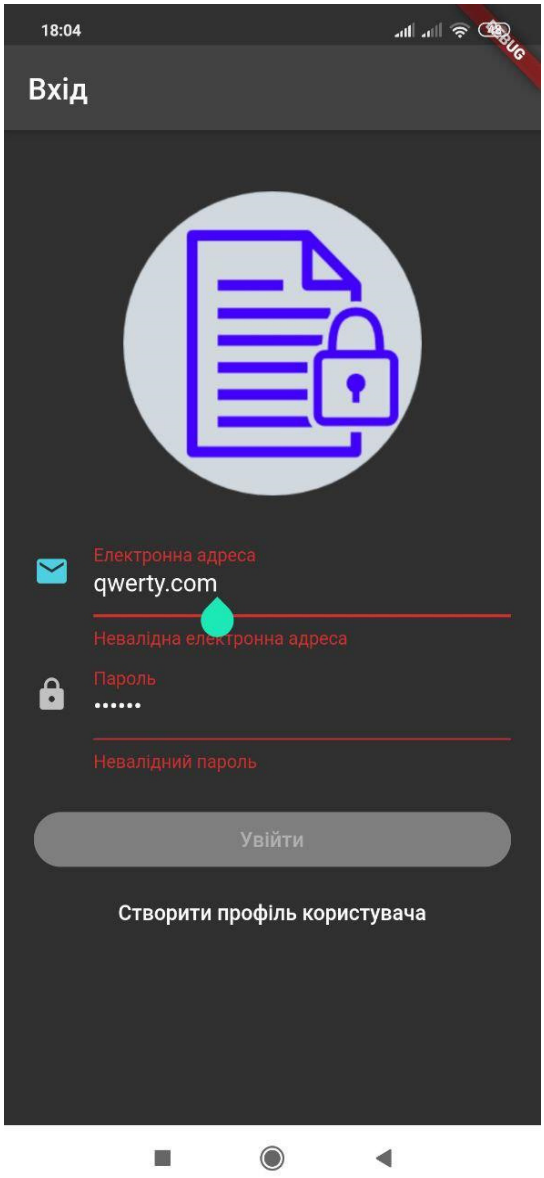


Рис. 2.1. Неуспішна валідація

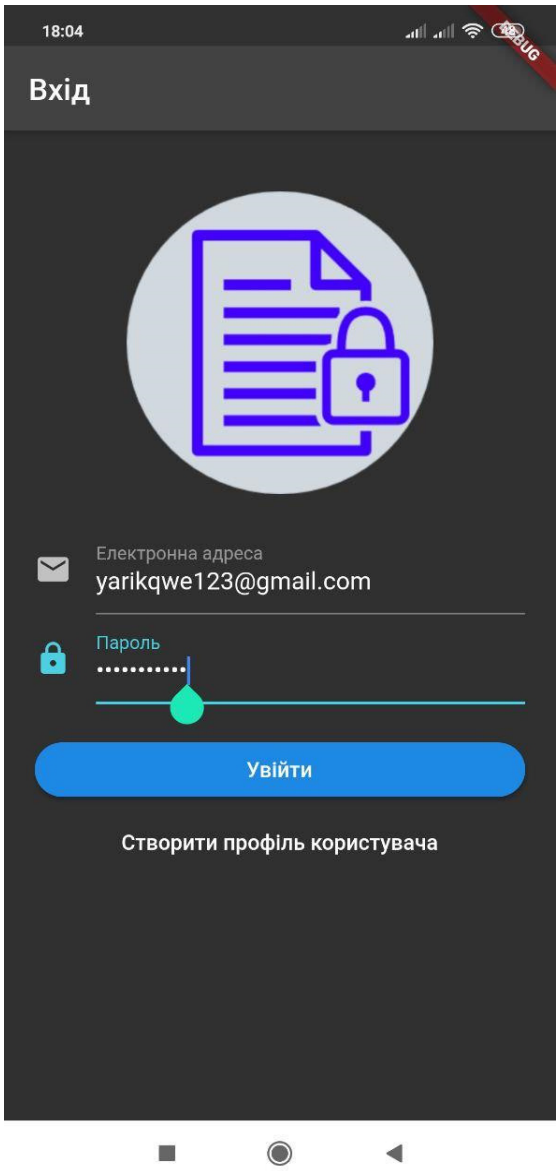


Рис. 2.2. Успішна валідація

У випадку, якщо користувач не має існуючого профілю, то йому необхідно зареєструватися у застосунку. Після переходу на екран реєстрації, відображається форма реєстрації. Вона містить поля для вводу електронної адреси та паролю користувача. Ці поля валідуються аналогічним способом, як і для форми автентифікації. Доки значення полів не стануть валідними, кнопка “Реєстрація” є неактивною. На рис. 2.3 та рис. 2.4 зображена форма реєстрації, що заповнена валідними та невалідними значеннями для полів вводу.

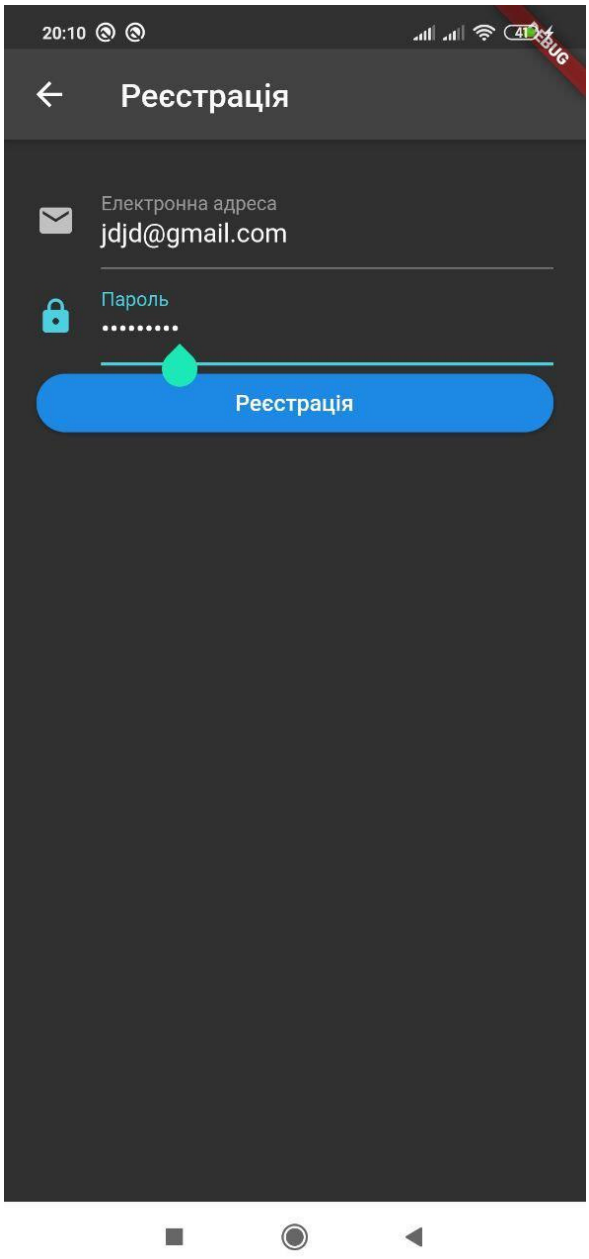


Рис. 2.3. Успішна валідація

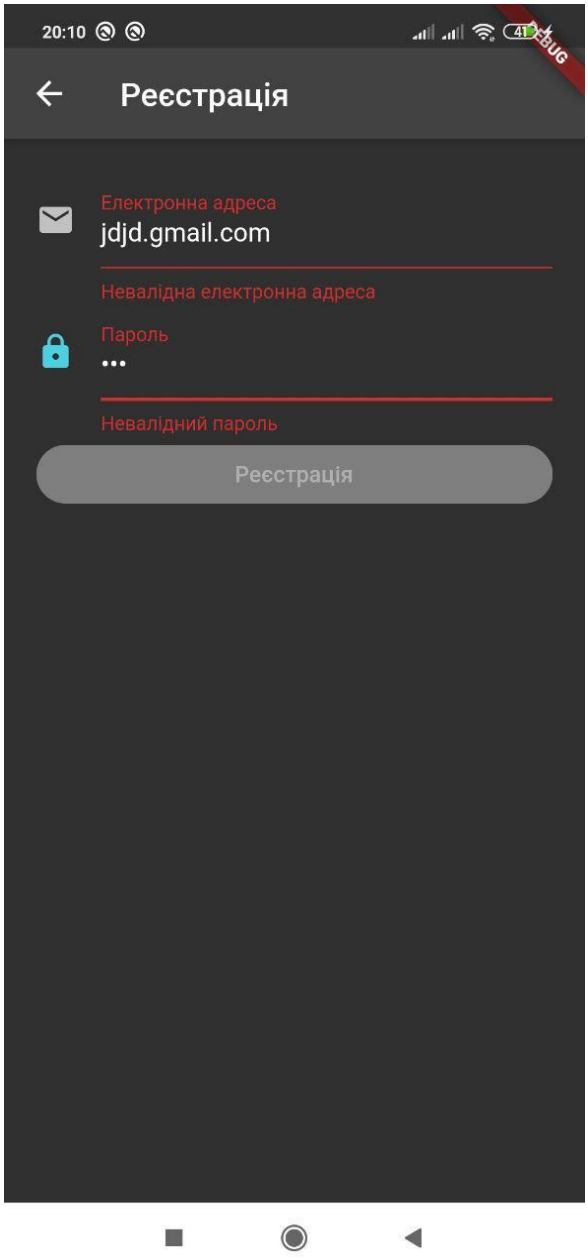


Рис. 2.4. Неуспішна валідація

3. Процедура управління папками та документами

Після успішної реєстрації або автентифікації, відкривається головний екран, що відображає список файлів та папок користувача. Головний екран має навігацію у вигляді табів, що розташовуються на нижній панелі екрану. Активна таба підсвічується додатковим кольором. Таба “Мої” відображає документи та папки, власником яких є авторизований користувач (рис. 2.5). У табі “Спільні” відображаються документи, якими інші користувачі поділились з авторизованим (рис. 2.6).

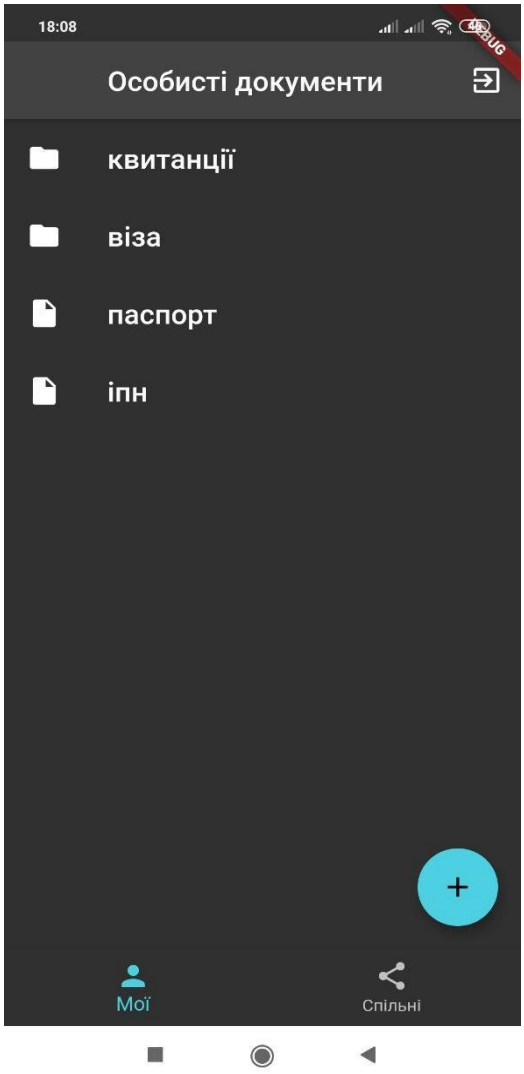


Рис. 2.5. Активна таба “Мої”

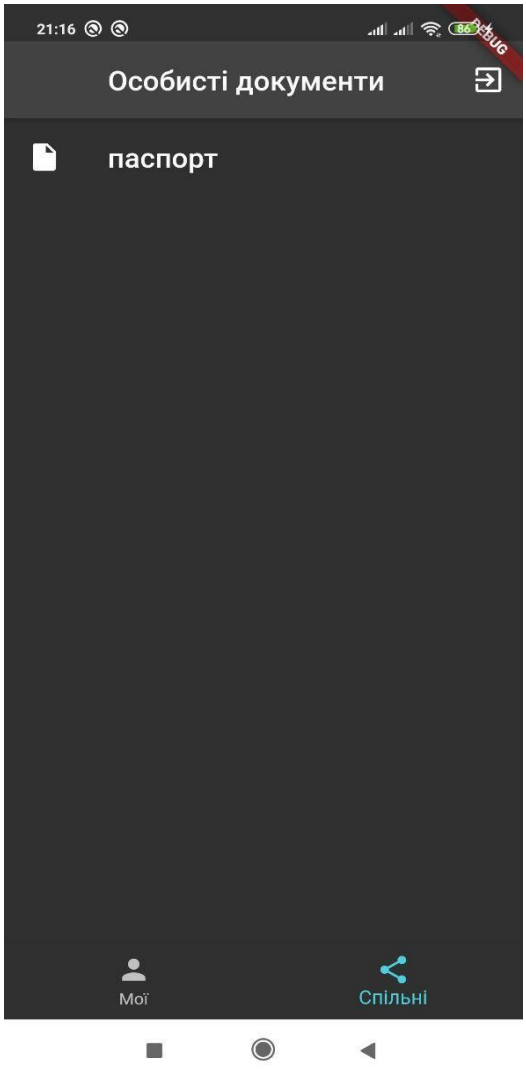


Рис. 2.6. Активна таба “Спільні”

Якщо таба “Мої” обрана активною, то над панеллю управління табами доступна кнопка з символом “+” для додавання нової папки, або нового

документа. При її натисканні замість панелі табів з’являється додаткове меню для вибору вікна створення: папки або документу.

Верхня панель головного екрану відображає назву поточної папки та має кнопку виходу з поточного профілю. У випадку, якщо це коренева папка, то панель містить текст “Особисті документи”.

Екран для додавання та редагування папки має форму, що містить поле для значення назви папки (рис. 2.7). Екран для додавання та редагування документу має форму, що містить поле для значення назви документу та динамічні поля для додавання будь-якої кількості довільних пар ключ-значення, що містять дані документу (рис. 2.8).

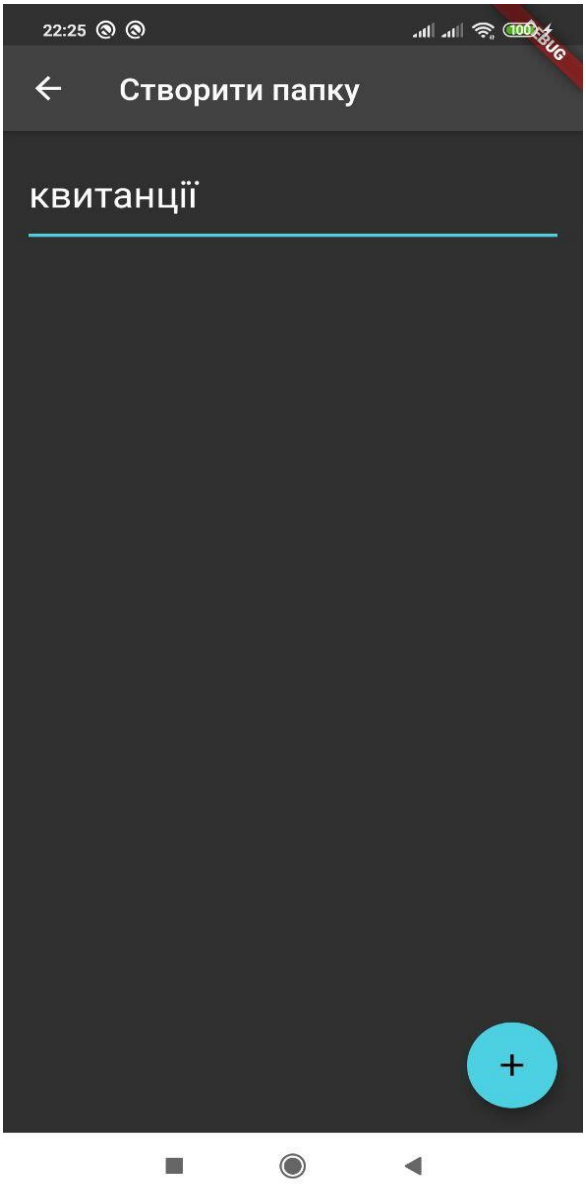


Рис. 2.7. Створення папки

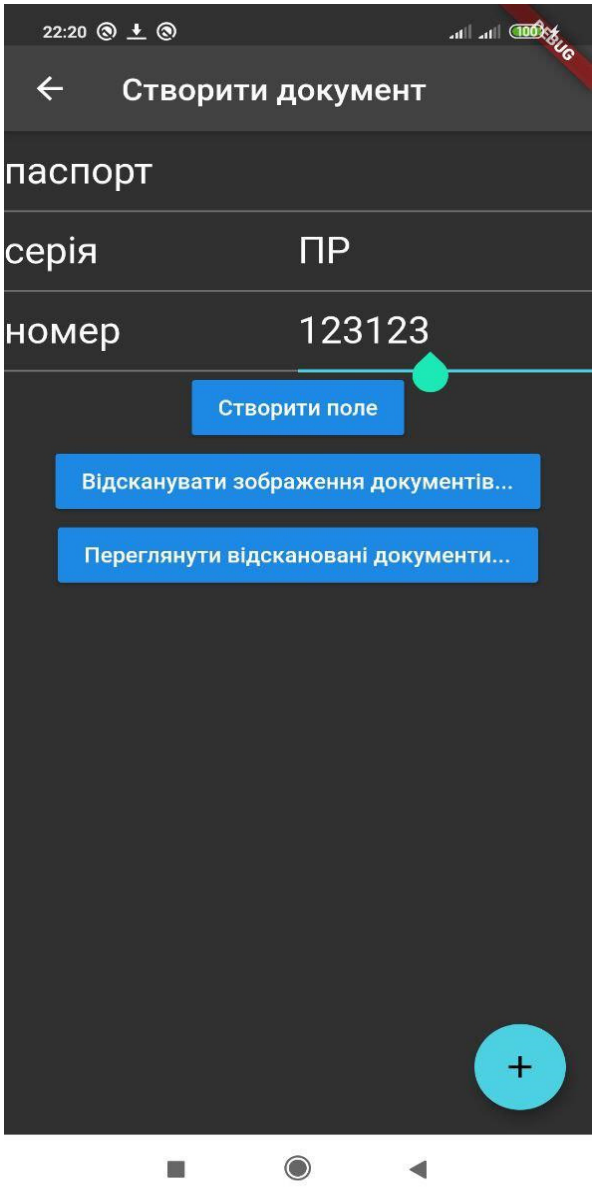


Рис. 2.8. Створення документу

Також екран документу містить кнопки для переходу на екрани сканування (рис. 2.9) та попереднього перегляду зображень документу (рис. 2.10). Екран сканування відображає поточне зображення з камери мобільного пристрою та зеленим прямокутником автоматично виділяє область, що була ідентифікована як документ. На екрані також з’являються підказки, якщо кут нахилення пристрою не дозволяє зробити чіткий знімок. Екран перегляду зображень показує поточні зображення документу у зменшеному форматі.

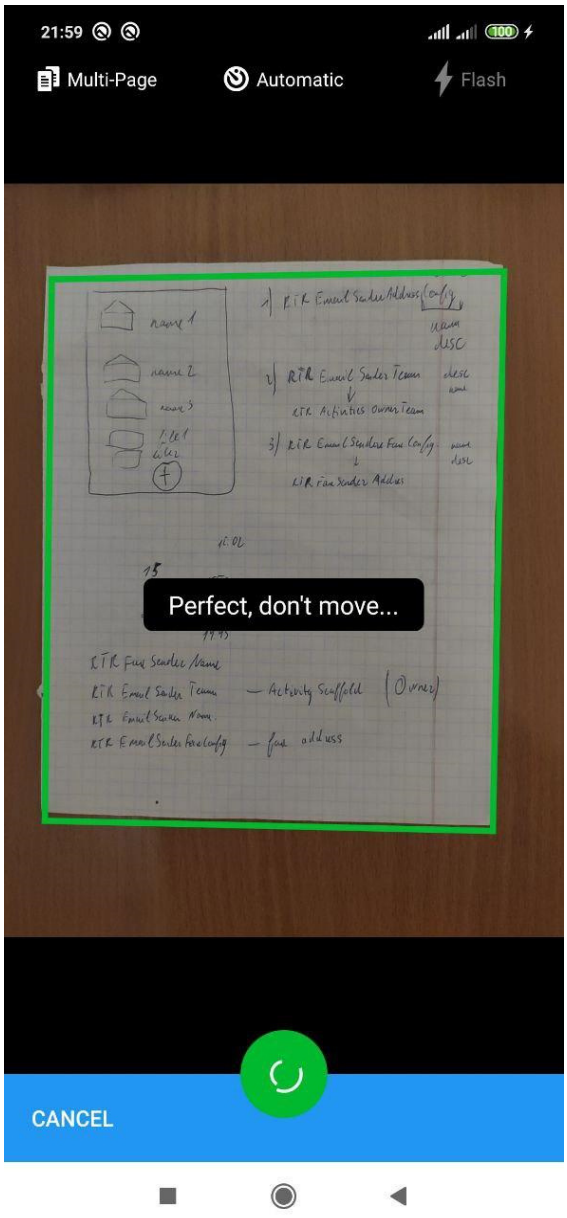


Рис. 2.9. Сторінка сканування

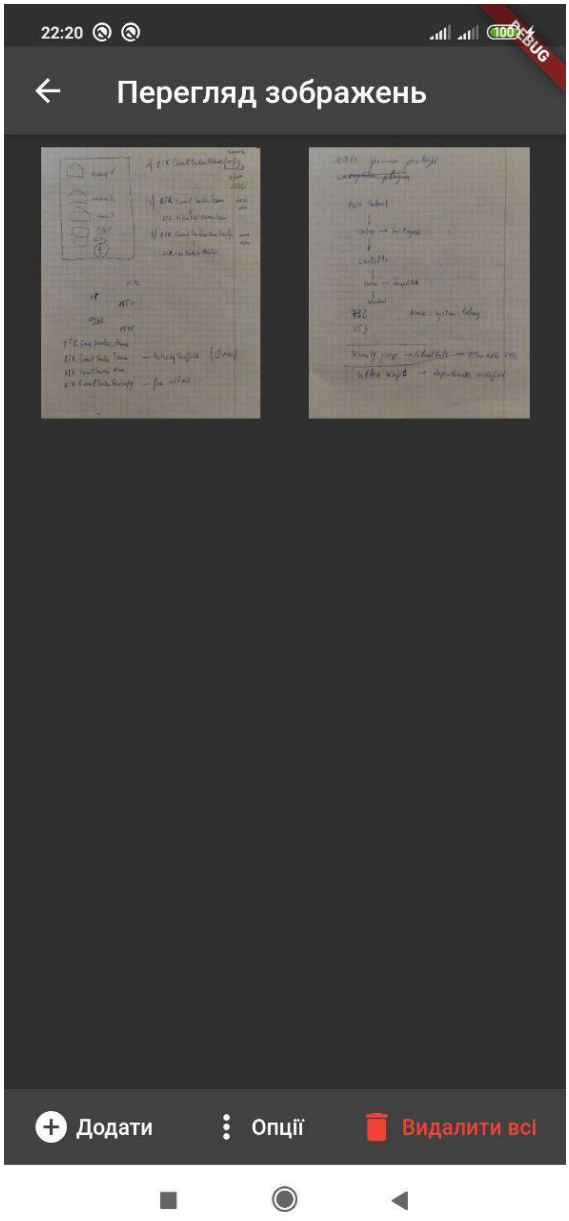


Рис. 2.10. Сторінка перегляду

Для того, щоб перейти в іншу папку необхідно натиснути на неї в списку головного меню. Після цього, в головне меню завантажується список файлів та папок, що знаходяться всередині обраної. Верхня панель відображає назву папки, в якій зараз знаходиться користувач. Для того, щоб редагувати, або видалити папку необхідно натиснути і потримати кілька секунд. Після цього з'явиться меню з опціями "Редагувати" та "Видалити" папку. Якщо користувач натискає кнопку "Редагувати", то відкривається екран з формою для редагування назви папки. Якщо користувач натискає кнопку "Видалити", то папка видаляється, разом з усіма іншими наявними всередині неї документами та папками.

Для того, щоб перейти на екран детального перегляду документу необхідно натиснути на нього у списку головного меню. Кнопка зі знаком "+", що розташована внизу екрану, дозволяє перейти на екран редагування документу. Також, на екрані детального перегляду у верхній панелі стає доступне меню з наступними опціями: "Видалити" та "Поділитися". При натисканні кнопки "Видалити" - документ видаляється. Якщо натиснути кнопку "Поділитися", з'явиться меню з полем вводу для електронної адреси користувача, з яким необхідно поділитися даним документом. Користувач, з введеною електронною адресою матиме у списку "Спільні" даний документ.

4. Процедура користування сервісом автозаповнення

Для того, щоб користуватися сервісом автозаповнення необхідно після встановлення мобільного застосунку обрати його у якості сервісу для автозаповнення у налаштуваннях мобільного пристрою (рис. 3.1).

Після активації сервісу автозаповнення, користувач має змогу використовувати дані, що пропонує сервіс, при заповненні електронних форм інших мобільних застосунків (рис. 3.2).

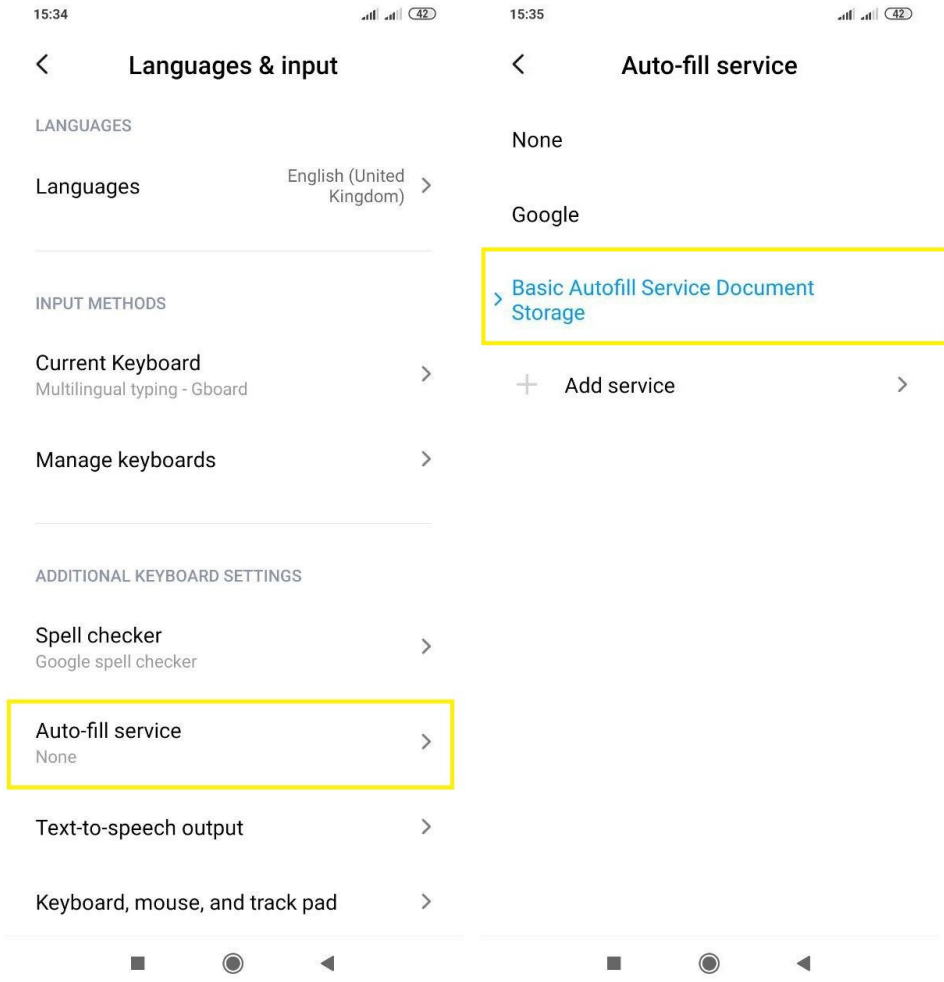


Рис. 3.1. Процедура налаштування сервісу автозаповнення

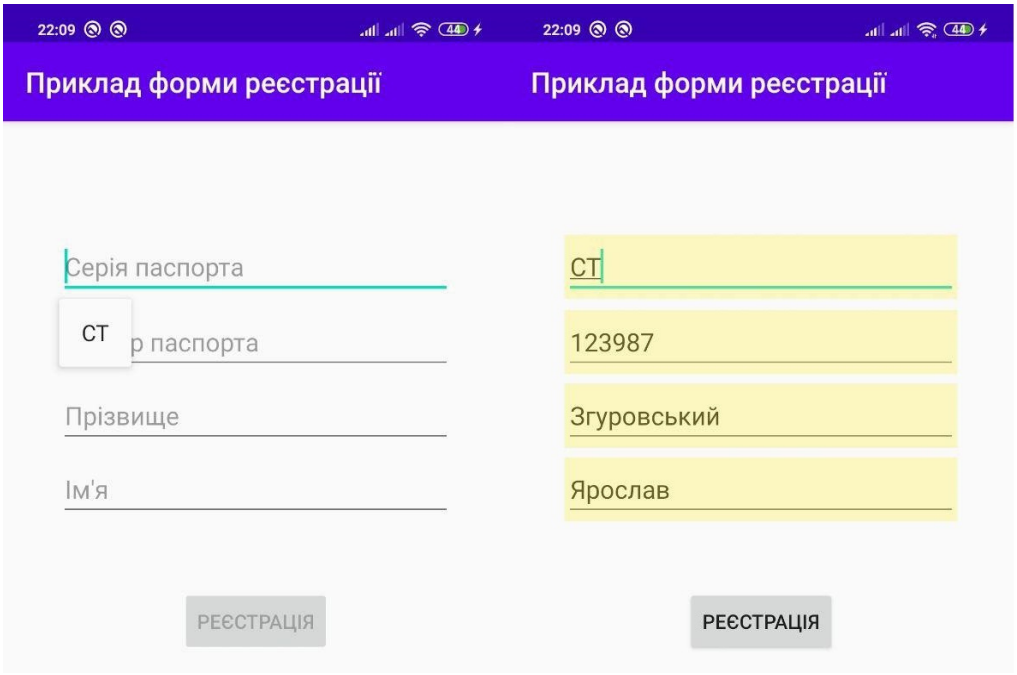


Рис. 3.2. Процедура автозаповнення форми даними із застосунку